April 1987

Report No. STAN-CS-87-1152

PB96-151592

Interpreting Natural Language Database Updates

by

James Edward Davidson

Department of Computer Science

Stanford University
Stanford, CA 94305

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188
Exp. Date: Jun 30, 1986

1a. REPORT SECURITY CLASSIFICATION unclassified	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release: distribution unlimited
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) STAN-CS-87-1152	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Department of Computer Science 6b OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Stanford University Stanford, CA 94305	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION DARPA 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00039-82-C-250
8c. ADDRESS (City, State, and ZIP Code) 1400 Wilson Blvd. Arlington, VA 22209	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Interpreting Natural Language Database Updates

12. PERSONAL AUTHOR(S)
James Edward Davidson

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
technical	FROM _____ TO _____	April 1987	119

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

Although the problems of *querying* databases in natural language are well understood, the performance of database *updates* via natural language introduces additional difficulties. This thesis examines the problems encountered in interpreting natural language updates, and describes an implemented system that performs simple updates.

The difficulties associated with natural language updates result from the fact that the user will naturally phrase requests with respect to his conception of the domain, which may be a considerable simplification of the actual underlying database structure. Updates that are meaningful and unambiguous from the user's standpoint may not translate into reasonable

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c OFFICE SYMBOL

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted.
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

19. Abstract (continued)
changes to the underlying database.

The PIQUE system (Program for Interpretation of Queries and Updates in English) operates by maintaining a simple model of the user and interpreting update requests with respect to that model. For a given request, a limited set of "candidate updates"—alternative ways of fulfilling the request—are considered, and ranked according to a set of domain-independent heuristics that reflect general properties of "reasonable" updates. The leading candidate may be performed, or the highest ranking alternatives presented to the user for selection. The resultant action may also include a warning to the user about unanticipated side effects, or an explanation for the failure to fulfill a request.

This thesis describes the PIQUE system in detail, presents examples of its operation, and discusses the effectiveness of the system with respect to coverage, accuracy, efficiency, and portability. The range of behaviors required for natural language update systems in general is discussed, and implications of updates on the design of data models are briefly considered.

Interpreting Natural Language Database Updates

By

James Edward Davidson

Abstract

Although the problems of *querying* databases in natural language are well understood, the performance of database *updates* via natural language introduces additional difficulties. This thesis examines the problems encountered in interpreting natural language updates, and describes an implemented system that performs simple updates.

The difficulties associated with natural language updates result from the fact that the user will naturally phrase requests with respect to his conception of the domain, which may be a considerable simplification of the actual underlying database structure. Updates that are meaningful and unambiguous from the user's standpoint may not translate into reasonable changes to the underlying database.

The PIQUE system (Program for Interpretation of Queries and Updates in English) operates by maintaining a simple model of the user and interpreting update requests with respect to that model. For a given request, a limited set of "candidate updates"—alternative ways of fulfilling the request—are considered, and ranked according to a set of domain-independent heuristics that reflect general properties of "reasonable" updates. The leading candidate may be performed, or the highest ranking alternatives presented to the user for selection. The resultant action may also include a warning to the user about unanticipated side effects, or an explanation for the failure to fulfill a request.

This thesis describes the PIQUE system in detail, presents examples of its operation, and discusses the effectiveness of the system with respect to coverage, accuracy, efficiency, and

portability. The range of behaviors required for natural language update systems in general is discussed, and implications of updates on the design of data models are briefly considered.

This research, within the Knowledge-Based Management Systems (KBMS) project, has been supported by ARPA contract N39-82-C-250. The views and conclusions in this document are those of the author and should not be interpreted as representative of the official policies, either expressed or implied, of the U.S. Government.

This is a revised version of a thesis submitted to the Department of Computer Science and the Committee on Graduate Studies of Stanford University in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Author's current address:

Teknowledge, Inc.
1850 Embarcadero Rd.
Palo Alto, CA 94303
Phone: (415) 424-0500

Acknowledgements

I would like to thank:

— Gio Wiederhold, my advisor, for his inspiration and support throughout this work,

— Jerry Kaplan, my mentor, for suggesting the topic and providing the time and guidance to help me turn it into a thesis,

— Jeff Ullman, for showing me the difference between science and hacking,

— the friends and colleagues, too numerous to mention, who made the graduate student experience so memorable,

— Avron Barr, Beth Bottos, Shel Finkelstein, and Dave Shaw, for providing so many insights on computer science and life,

— the agencies who funded this work: the National Research Council of Canada, the Defence Advanced Research Projects Agency, and Teknowledge, Inc.,

— Linda DeMichiel, for the support and encouragement she has given me,

— my parents, for always being there.

Table of Contents

Chapter 1
Introduction

Natural language is a desirable access mechanism for database systems because it frees the user from the task of understanding the details of the database structure. A number of systems have provided natural language *query* capabilities; few of these, however, allow the user to perform *updates* to the database using natural language.

The provision of update capabilities introduces problems not encountered in processing queries. These difficulties stem from the fact that users will naturally phrase requests with respect to their conception of the domain, which may be a considerable simplification of the actual underlying database structure. Updates which are meaningful and unambiguous to the user may not translate into reasonable changes to the underlying database. Update requests may be *impossible* (cannot be performed in any way), *ambiguous* (can be performed in several ways), or *disruptive* (can be performed only in ways which cause undesirable side effects). Different mappings of an update request into the actual database structure may result in different effects. Some of these effects may be undesirable or unanticipated. While human speakers would intuitively reject these unusual readings, a computer program may be unable to distinguish them from more appropriate ones.

For example, the simple request "Change the teacher of CS345 from Smith to Jones" might be carried out by altering the number of a course that Jones already teaches to be CS345, by changing Smith's name to be Jones, or by modifying a "teaches" link in the database. While all of these may literally carry out the update, they may implicitly cause unanticipated changes such as altering Jones' salary to be Smith's.

This dissertation describes an approach in which an update is treated as a request to put the database into a self-consistent state in which the request is satisfied. The problem is

then to select the most desirable of (potentially) several such states. This is the state that is the "nearest" one to the current state, in the sense that it involves the least disruption of the database. A set of domain-independent heuristics is used to rank the potential changes to the database. This ranking process may be guided by various linguistic considerations, such as the difference between "transparent" and "opaque" readings of the user's request, the distinction between the "sense" and "reference" of referring expressions, and the interpretation of counterfactual conditionals.

The approach taken here is motivated by considerations of portability and efficiency. No domain-specific knowledge is required beyond that typically provided in database systems. Furthermore, the method is designed to minimize references to the database extension, relying instead on information in the database schema and the structure of the user's request.

This approach has been implemented in a system, PIQUE (*Program for Interpretation of Queries and Updates in English*), which is described. PIQUE operates by retaining a simple model of the user's view and interpreting updates with respect to that model. For a given update, PIQUE considers the possible ways of fulfilling the request and ranks them. The appropriate candidate is executed. If no candidate is clearly superior to the others, the system engages in a clarification dialogue with the user.

Chapter 2 ("Motivation and Approach") explains the natural language update problem in more detail, intorduces the PIQUE system, and outlines the issues addressed by the thesis. Chapter 3 is an extended presentation of the PIQUE mechanism. Chapter 4 presents a number of examples of the system in operation. Chapter 5 is an evaluation of PIQUE with respect to coverage, correctness, portability, and efficiency. Chapter 6 surveys related work and presents two alternative approaches to the update problem. Chapter 7 contains the conclusions. Readers wishing only a cursory understanding are advised to read Chapters 2, 5, 6, and 7, and Section 3.3 ("Ranking Candidate Translations") and to skim the rest.

Chapter 2
Motivation and Approach

2.1. The Problem

To show some of the problems that can arise during updates, we consider a relational database of employees, salaries, departments, and managers. The database consists of two relations:

```
ED:                          DM:
EMP     SAL    DEPT          DEPT     MGR
-----------------            ------------
Adams   30     Invntry       Sales    Jones
White   35     Mkting        Mkting   Baker
Brown   25     Sales         Invntry  Fisher
Smith   30     Sales
Palen   25     Sales
```

The following is a dialogue between a user and the system; comments are enclosed in brackets.

Example 2.1:

```
Q1: List the employees and their managers.

R1:     EMP     MGR
        ------------
        Adams   Fisher
        White   Baker
        Brown   Jones
        Smith   Jones
        Palen   Jones

Q2: Change Brown's manager from Jones to Baker.

R2: Done.
```

[Apparently the system has fulfilled the user's request.]

```
Q3: What is the average salary paid to Jones'
```

```
        employees?

R3: $0.

Q4: List Jones's employees.

R4: NIL
```

[From these responses, the user realizes that something has gone wrong.]

```
Q5: List the employees and their managers.

R5:     EMP     MGR
        ------------
        Adams   Fisher
        White   Baker
        Brown   Baker
        Smith   Baker
        Palen   Baker
```

[The user sees that the system has made two unanticipated changes—changing Smith's and Palen's managers—in addition to the one that was requested.]

From the user's point of view, his request is meaningful and unambiguous. He sees a set of values and asks to change one of them. We cannot expect the user to know that employees and managers are linked via their departments. The problem lies in the fact that the update request can be performed in two ways:

(a) by making Baker manager of the Sales department;

(b) by moving Brown from the Sales department to the Marketing department.

Both of these changes literally fulfill the request. The system, lacking any means for deciding between them, has apparently chosen (a), making Baker the manager of the Sales department. This change, however, has the unanticipated effect that two other employees have had their managers changed. Furthermore, the system has failed to alert the user to these extraneous changes.

2.2. Alternative Characterizations of the Problem

Two alternative formulations of the problem, one deriving from work in database management and the other from the philosophy of language, provide additional insight.

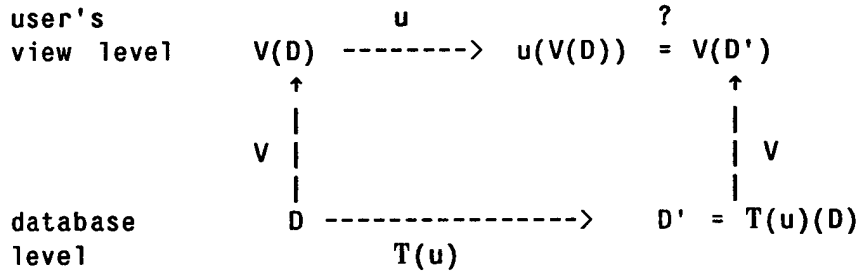
2.2.1. Updates to database views

Given a database structure, define the user's *view function* V as the transformation that is applied to the database to yield the conceptualization with which the user works. In Example 2.1 the view function, as defined by Q1, is a transformation consisting of a join and a projection, which is applied to the original two files to yield a single new file with only two attributes. Define the user's *view* as the result of applying the view function to a given state of the database; in the example, this produces a file with five entries, as was shown in R1.

A user's update request, **u**, is a request to update the view. In the example, the request is stated in Q2. Since the view is only "virtual" (derived from the data), it can be modified only indirectly, by means of changes to the underlying database. Call the result of translating the update request to the database level **T(u)**. The objective is to find the change to the underlying database that comes closest to having the desired effect on the user's view. That is, the desired translation **T(u)** is one which produces a revised database such that, when the view function is applied to that database, the result is the view requested by the user.

In graphical terms:

> **D** represents the initial state of the database, **D'** the state that results after applying the translated update **T(u)**; **V(D)** and **V(D')** are the views that result from applying the view function V to the database states.



In mathematical terms, the mapping V from the underlying database **D** to the user's view **V(D)** induces a *homomorphism.* Loosely, a homomorphism is a function that preserves the structure of its arguments under given operations. In this case, the operations are the

changes to the underlying database and the corresponding changes to the user's view. The difficulties with updates expressed on the view (rather than on the underlying database) arise from the characteristics of the inverse of this homomorphism: elements in the user's view (states of the "conceptual" database) map under $\mathbf{V^{-1}}$ into a set of states of the underlying database.

If the mapping V is invertible, i.e. if $\mathbf{V^{-1}}$ is also a function, then an *isomorphism* is induced. In this case, each requested update will have a single, unambiguous translation in the underlying database, and the difficulties discussed here do not arise. In general, however, this is not the case. The set of database states produced by $\mathbf{V^{-1}}$ may be empty, if the view update cannot be accomplished in any way; in this case, the request is said to be *impossible.* The set may instead have many elements, in which case the request is ambiguous with respect to **D**, and is said to be *underspecified.*

The ideal update translation will produce a database state D′ which, when transformed by the user's view function, yields exactly the revised state that he requested. Our method considers a broader class of changes to the database that literally fulfill the user's request but may not yield precisely the intended view **u(V(D))**. In the example, there were two translations of the user's request; update (b) yielded the exact view, update (a) a different one.

This characterization of the problem is identical to that used in database management in addressing the problem of updates to *views.* In database terminology, a *view* (or *external relation*) is a transformation of the database, specified by the database administrator, which defines the aspects of the database accessible to the user. The view update problem has been extensively investigated by database researchers; see ([Dayal and Bernstein, 1982], [Bancilhon and Spyratos, 1982], [Chamberlin, Gray and Traiger, 1975], [Keller, 1982]). There are two essential differences between database view updates and the natural language updates presented here:

(a) In formal database work, views are fixed and specified in advance, not inferred dynamically as in our work;

(b) Updates through database views are allowed in only a limited set of cases—those in which there is a unique candidate update; no attempt is made to handle ambiguous cases, where multiple candidates exist. (See [Keller, 1982], [Masanaga, 1983] for slight extensions of this.)

The formal view update work, and its relationship to our work, will be discussed further in Sections 3.2, 3.3, and 6.2.

2.2.2. Counterfactuals

If a natural language update request is ambiguous—i.e., if multiple candidate translations exist—it is necessary to choose among these candidates, selecting the appropriate one for execution. In general, this choice must be made on the basis of information external to the request.

In this respect, the natural language update problem is related to the *possible worlds* work of Lewis, developed formally in his book *Counterfactuals* [Lewis, 1973]. In this work, Lewis examines the meaning and formal representation of such statements as "If kangaroos had no tails, they would topple over". He argues that to evaluate the correctness of this conditional statement, and similar counterfactual conditionals, it is necessary to construct in one's mind the possible world minimally different from the real world in which the condition would be fulfilled (the "nearest" consistent world). He points out that this hypothetical world not only differs in that kangaroos do not have tails, but also reflects other changes which would be required to make the world plausible. Thus he rejects the idea that in the hypothetical world kangaroos might use crutches (as not being minimally different), or that they might leave the same tracks in the sand (as being inconsistent).

Lewis uses more precise terminology: a world i has associated with it a (possibly infinite) set S_i of worlds which are *accessible* from i. A comparative similarity relationship $\leq_i$ defines the degree of similarity between other worlds and i. The $\leq_i$ relationship satisfies a number of properties, including transitivity and strong connectivity. The function $f(\varphi,i)$ selects, for any sentence φ and world i, the closest set of worlds to i in which φ is true; this is used in the evaluation of counterfactuals.

The application of this work to processing natural language database updates is to regard each transaction as presenting a "counterfactual" state of the world, and request that the "nearest" reasonable world (i.e., database state) in which the counterfactual is true be brought about. For example, the request "Change the teacher of CS345 from Smith to

Jones" might correspond to the counterfactual "If Jones taught CS345 instead of Smith, how would the database be different?" along with a speech act requesting that the database be put in this new state. To select this nearest world, the candidate updates are evaluated with respect to a metric which heuristically estimates the conceptual magnitude of the change. A major component of this metric is determination of the number and type of side effects induced by each candidate. Side effects that disrupt the user's view are considered more "severe" than changes to portions of the database not in that view. The update that involves the least change is selected as the most appropriate. This subject is discussed in detail in Section 3.3.

Updates that violate syntactic or semantic constraints implicit in the database structure and content are eliminated as *inconsistent.* Functional dependencies and subset constraints, in particular, are useful semantic filters.

In general, an update request may have an infinite number of associated candidates, most of which consist of manipulation of "dummy" values and links. Most of these are counterintuitive and are eliminated a priori by restrictions on the kinds of changes allowed to the database in response to a request. (See Section 3.2 for details on this.) In Lewis' terms, this amounts to a restriction on the *accessibility* relationship between possible worlds.

2.3. The PIQUE System

We have designed and implemented a system PIQUE (*Program for Interpretation of Queries and Updates in English*), which processes natural language updates. The approach used in PIQUE is summarized here and presented in detail in the next chapter.

Our approach to update interpretation involves four phases:

(1) maintain a model of the user;

Since the user will phrase update requests with respect to his view of the database, it is important to have some model of that view. The user view is necessary for determining both the intended effect of the update (i.e., which database entities the user wants

changed), and the appropriate method of carrying it out, as well as the appropriate level of informativeness for responses and explanations. In PIQUE, the view is assumed to be the segment of the database currently being examined, as defined by the most recent queries.

(2) given an update request, generate the set of candidates that fulfill the request;

In general there will be many ways of changing the database which produce at least the desired effect. This set of candidate translations may be large or small, depending on the definition of "acceptable" translations and "performance" of updates. This set can be generated algorithmically, using the view definition, the update request, and, if necessary, the current database extension.

PIQUE considers only candidates that meet certain restrictions concerning the relationship between the request and the translation. This eliminates most non-intuitive translations, and yields mainly candidates that correspond to different readings of expressions in referentially opaque contexts [Quine, 1961].

(3) rank the candidates;

The resulting candidate translations may be evaluated with respect to their "reasonableness" as a method of performing the update. This ranking may use information from five sources:

- –consideration of the side effects associated with each translation;
- –semantic descriptions of the changes that occur in the domain;
- –integrity constraints contained in the database schema;
- –pragmatic information indicating the "magnitude" of the translations;
- –consideration of the number of changes entailed by each translation.

The PIQUE system will use any information which is available, including all the above types, but does not require the provision of any knowledge beyond that regularly found in database systems. This domain-independence and the resultant *portability* are key factors in the design of PIQUE.

(4) take action;

The action to be taken is determined by the results of the ranking. Depending on the number of resulting candidates, and their differentiation by the ranking process, the appropriate response may be performance of one of the candidates, presentation of one or more candidates to the user for confirmation or selection, notification of possibly unanticipated side effects, or explanation of failure.

PIQUE uses an algorithm under which the leading candidate is performed if it outranks the others by a sufficient amount. If a number of candidates have equal ranking, they are presented to the user for a choice. If the system is unable to fulfill the request (typically because of integrity constraints in the database), the failure is explained to the user.

2.4. Issues Addressed

Within the context of the general problem of natural language updates, this thesis addresses a number of issues in both artificial intelligence and database management. We present here a brief list of the issues; these are discussed further during the technical presentation, evaluation and conclusions (Chapters 3, 5 and 7 respectively).

2.4.1. User modeling

A number of artificial intelligence systems have relied on models of the user for various tasks. PIQUE maintains a simple user model which is used for two purposes:

(a) determining the virtual database structure with respect to which the user has presumably phrased his request;

(b) deciding which information is appropriate for inclusion in a response.

2.4.2. Nonliteral responses

As discussed in Kaplan's thesis [Kaplan, 1979], there are situations in which a literal response from a database system is not appropriate (specifically, questions with failed *presumptions*). This thesis examines another class of such situations, notably update requests. A response of "yes" or "no" to an update request may be uncooperative; more appropriate behavior may consist of a warning about side effects or an explanation for a failure.

2.4.3. Natural language generation

Many database systems have extensive capabilities for interpretation of natural language requests. Less consideration has been given to the generation of natural language responses, in part because the range of required responses has generally been limited. PIQUE exhibits a simple capability for language generation, tailored to the responses needed in update situations.

2.4.4. Selecting among multiple possible states

Lewis [1973] suggests the use of possible worlds and a distance metric, for evaluating counterfactuals, but provides little information about what form such a metric might take. This thesis describes an implemented set of syntactic and semantic heuristics which provide such a metric for database states, and discusses the behavior induced.

2.4.5. Updates to database views

Work on database view updates has traditionally considered only situations in which the request is meaningful and unambiguous. This thesis examines a broader set of update cases and presents an algorithm that provides a response in all cases.

Chapter 3
The PIQUE System

In this chapter we present the details of the PIQUE system. The motivation for many of the technical aspects described here is discussed in Chapter 5, "Evaluation".

3.1. The User Model

Users of natural language database systems phrase requests with respect to the perceived structure of the domain and the context established by the dialogue, rather than the actual database. To ensure correct interpretation of those requests, the system must retain some form of model of the user.

For example, consider the following dialogue between a user and a database management system:

Example 3.1.

```
Q1: Who are the programmers?
R1: Jones, Smith, Baker, Schmidt.
Q2: Transfer Jones to the marketing department.
R2: There are 37 employees named "Jones";
        which one do you mean?
```

The system here is being uncooperative, failing to recognize the apparent intent of the user's second request. This problem occurs because the system attempts to interpret the request in isolation; on this basis, the referring noun phrase "Jones" in Q2 is genuinely ambiguous. However, the context of the first question/response pair strongly suggests a likely referent, and this should be detected by the system.

The use of the abbreviated form of reference by the user for his second request is not an isolated occurrence. Users of "intelligent" systems will tend to attribute human-like intelligence to those systems. In the case of natural language systems, this means that the

users will observe some of the rules of conversational coherence and phrase inputs with respect to the current context. In the example above, the user has obeyed the "Cooperative Principle" [Grice, 1975] and made his specification exactly as informative as (he believes to be) necessary. To prevent the kind of failure which has occurred here, the system must retain some model, however simple, of the user's current "state of mind."

3.1.1. User modeling in artificial intelligence and database management

Classes of user models for computer systems can be divided along three dimensions [Rich, 1979]: *explicit/implicit* (does the user have to build the model himself, or does the system infer it from his behavior), *canonical/individual* (is the model intended to represent users in general, or is there a different model for each user), and *long-term/short-term* (is the model intended to represent the user's profile in general, or is it intended to be dynamic, changing as he focuses on different tasks). With respect to this framework, the class of models considered here is implicit and individual. The PIQUE model represents both long-term characteristics (the user's beliefs about the structure of the database and domain), and short-term ones (the part of the database on which he is currently focusing). The difference between these two aspects is not sharp, and they are represented together in the model.

There has been some work in artificial intelligence on the use of such models in dialogues, although none in the context of database access. Grosz [1977] developed a system for representing focus in task-oriented dialogues. That model relied on a domain-specific representation of the *task hierarchy*—the global relationship between the tasks and sub-tasks being worked on. The focus model (which was represented using partitioned semantic networks) was used mainly to resolve non-pronominal definite noun phrases appearing in the discourse. Sidner [1979] described a more general method of anaphora comprehension, which relied on a model of focus similar to Grosz's. Cohen and Perrault [1979] developed a sophisticated model, capable of representing the user's wants, beliefs, and intentions, as part of a plan-based theory of speech acts.

Database interactions present particular problems for implicit modeling, because there is in general no *a priori* structure to the kinds of dialogues which can occur, as there is, for

example, in task-oriented dialogues. The user's access of the database will not necessarily follow any global pattern.

Within database management, only restricted forms of user modeling have been provided. Database theory provides a formal notion of an *external model.* The external model is a transformation of the conceptual model (the underlying database), representing the aspect of the database visible to a user or group of users. The external model is composed of *views* (derived relations), formed from the underlying database by operations such as *projection* (e.g., a user may be allowed to examine employee records, but not the salary attribute), *selection* (e.g., the user may examine the records of only those employees in the sales department), or *join* (e.g., the user may examine records of employees and their associated departments). However, the structure of the external model is long-term and must be fully specified by the database administrator in advance of any transactions involving the model.

Implicit models have been considered in only a few instances. Rowe [1982] has implemented a system that takes into account the user's *preferences,* as dictated by the task that he is trying to accomplish, to decide which items from the database to present, and in which order to present them. However, this depends upon (a) having a pre-specified model of the task to be performed, and (b) recognizing the current dialogue as an instance of an attempt to perform that task. Finkelstein [1982] uses a short-term, implicit, individual model, to improve efficiency of query response by recognizing commonalities between successive queries. While his mechanism is similar to ours, his goal—optimization of response time—is very different from ours, which might be termed *cooperativeness* or *habitability.*

In what follows, we will use the term *view* or *view model* to describe the PIQUE user model, because of the similarity—in structure and function—to the mechanism of the same name in the database domain. It differs from the database version in its implicit derivation, its inclusion of short-term (focus) information, and its dynamic nature.

3.1.2. View models and database updates

The importance of a view model in the interpretation of an update request is that the *meaning* of such a request may vary according to the view. Since the user will phrase his requests with respect to his current view (as established by the dialogue), the context of the dialogue must be used during interpretation of the request to discover the user's *intent.*

The dialogue context may affect the meaning of a natural language update in at least two ways. (Note that some of these contextual phenomena may also arise in queries; we will consider only those aspects that pertain to interpretation of updates.)

(a) The set of entities known to the user may be restricted.

This phenomenon occurs in Example 3.1 at the start of this section. The user focuses on a subset of the set of employees and then asks for an update. The phrase "Jones" is ambiguous with respect to the total database, but can be meaningfully interpreted in the context of the previous query. In this case, the view has served to restrict the set of entities that are to be affected by the update.

(b) A particular path through the database between two relations may be preferred, even though a number of such paths exist:

Example 3.2:

```
Q1:   Which graduate students are supervised by
          tenured professors?

R1:   Student   Professor
      Adams     Ashworth
      Nixon     Choate
      Peries    Moseley
      Devlin    Choate
      . . .

Q2:   Change Adams' professor to "Baer".
```

The term "professor" in Q2 could reasonably refer to the professor that Adams works for (or the one that he takes classes from, or . . .). None of the paths linking "Adams" and "professors" is preferable to the others. In the context of Q1, however, the user is apparently referring to the professor who supervises Adams; he may not even be aware of

the others. Here, the view has determined which path is to be used, hence which of Adams' "professors" is to be changed.

(c) The view model is used to discover the user's intent—what he wants to happen. The view may also influence the selection of the appropriate method of fulling the request, by indicating which attributes and relations the user is aware of. Consider a slightly modified version of the dialogue in Example 2.1:

Example 3.3:

```
Q1:   List the employees, their departments, and their managers.
R1:   Emp      Dept     Mgr
      Adams    Invntry  Fisher
      White    Mkting   Baker
      Brown    Sales    Jones
      Smith    Sales    Jones
      Pullum   Sales    Jones
Q2: Change Brown's manager from Jones to Baker.
```

The technique of moving Brown to a new department is no longer as appropriate as in Example 2.1; since the user is aware of the department, an attempt to change Brown's department would probably have been phrased differently.

3.1.3. The PIQUE Model

The PIQUE system relies on a user model which is implicit and individual. In brief, PIQUE models the user by noting which aspects of the database he has previously examined.

The development of a view model entails four considerations:

(a) a representation for the user's current view;

(b) a method of *deriving* the view representation and maintaining the model during the course of a dialogue;

(c) a basis for deciding when use of a particular view is appropriate in processing later inputs;

(d) an explicit mechanism for *using* the view representation.

The last aspect, use of the view model, forms the major part of this thesis, and is considered starting in Section 3.2; the first three points are discussed here.

The PIQUE view model can also be used to provide support for interpretation of queries, especially those containing definite noun phrases (see [Davidson, 1982]). The methods of using the view model in such cases are different from the ones used in interpreting updates.

3.1.3.1. The view representation

The user's queries to the database are expressed, at some level, in a formal data manipulation language (DML), which is typically a variant of the relational calculus or relational algebra [Ullman, 1980]. For example, the query "Who are the programmers?" might be expressed, in an idealized calculus-based DML:

```
{ xεEmployees | x.occupation="programmer" : x.name }
```[1]

i.e., "Print the names of all the members of the employees relation whose occupation is 'programmer'."

Besides being a query, this expression can be viewed as an *intensional description* of the class of programmers. Thus, the DML can serve as a representation language for describing segments of the database—a representation language that has the benefit of a formal semantics. Since the user's view is assumed to be that segment of the database that he is currently accessing, the DML is a suitable language for representing the view. In PIQUE, the current view, at any time, is represented intensionally by a DML expression. The *intension* (description) is a more useful concept than the *extension* because it describes, not only the entities that are currently in the view, but also the *aspect* of them that is currently of interest.

The view expressions used by PIQUE correspond to *conjunctive* queries—queries in which the qualification is a conjunction of equi-selection terms (e.g., `x.dept = sales`) and join terms (e.g., `x.dept = y.dept`). The views are further restricted to be *chain-structured*, in that the set of joins in the view does not form a cycle, no relation is involved in more than two joins, and no relation has more than one variable ranging over it. This is a limited, yet large class of views. Justification for these restrictions is discussed in Section 5.1.

---

[1] This DML will be used for presentation purposes throughout the thesis. The actual DML used in PIQUE is a modified form of SODA, a LISP-compatible variant of relational calculus developed by Robert Moore; except for syntax, it is fundamentally identical to the DML used here.

PIQUE views must be submodels of the database schema, in that all joins must correspond to *connections* specified in the database schema. The schema used by PIQUE is encoded in the structural model [Wiederhold and El Masri, 1980], an extended form of the relational data model. Besides encoding the relations and attributes as in the standard relational model, the structural model permits specification of connections between relations. These connections, which are represented by joins, serve to model semantic relationships between collections of entities. Thus, joins in the view must correspond to "natural" relationships.

There is an interesting correspondence between the *structure* of the view expression and the effects of the view. *Selections* in the view expression correspond to restrictions in the entities known to the user, as in Example 3.1. *Joins* in the view expression determine the path used for navigation, as in Example 3.2. *Projections* in the view expression indicate the user's awareness of specific attributes or relations, as in Example 3.3.

3.1.3.2. Deriving the view representation from the dialogue

PIQUE uses a simple method of tracking the view in a dialogue. Each request by the user establishes a temporary "view space", represented by the DML form of the request. Successive inputs may make use of this space or shift to a new space. View spaces are "stacked" as the dialogue progresses, to allow reference to previously mentioned events or entities.

Unlike the task-oriented domains used in Grosz's work, the database domain does not provide strong clues for the closing of a view space. Rather, such action is indicated by a shift to a new view space. Such shifts are indicated by queries which examine different areas of the database or invoke entities outside the previous space.

This method of tracking the dialogue by recording the query expressions is simple, yet adequate for many cases. If desired, more sophisticated rules could be introduced. For example, view spaces might be allowed to evolve incrementally over the course of a dialogue.

3.1.3.3. Deciding when use of a view model is appropriate

When an update request is received, it is matched against the view models on the stack, most recent first, to see whether one of these models can guide the interpretation of the update. The match test checks compatibility of attributes and paths. An update is compatible with a view model if:

(a) the attributes mentioned in the update request all appear in the target of the expression defining the view;

(b) the ranges of selections specified in the update are not mutually exclusive with the ranges of selections in the view definition.

These conditions ensure that the update can meaningfully be interpreted as applying to the domain established by the view model. Again, more sophisticated conditions could be established if desired. Some conditions for use of a view model are discussed in [Davidson, 1982].

Once the correct view model is established, the view expression and update request are passed on to the next phase of the system, which considers possible ways of fulfilling the request. In some cases the update request may not be compatible with any of the stored views, or there may not be any models stored. This would occur if the user requested an update without examining the appropriate segment of the database—for example, if he were working from a printed copy, or thought he knew the contents of the database. In such cases, it is often possible to infer the view from the update request alone. Thus, "Change Brown's manager . . ." might be meaningful—and in fact would be understood by PIQUE—even in the absence of preceding dialogue.

3.2. Generation of Candidate Translations

Given an induced view, and an update request to be interpreted with respect to that view, the system must generate the set of *candidate translations*, which change the underlying database in ways that fulfill the user's request. The problem is that there may be several translations which will produce the desired effect on the view. In this case, the update is *underspecified*; this is a common characteristic of natural language updates.

There will often be many non-intuitive translations which, while literally correct, are

semantically anomalous. For example, the request in Example 2.1 may be fulfilled by firing Brown and inserting another employee named Brown in Baker's department. This is presumably not what the user intended. In the PIQUE system, such anomalous translations are ruled out by a series of restrictions, discussed in Subsection 3.2.5

In this section, we first discuss the linguistic motivation for our approach to candidate generation. We then define the range of updates accepted and specify the restrictions on "acceptable" translations. Finally, we present the algorithm for generating translations and show that it finds all acceptable translations.

For both generating and ranking the translations, PIQUE uses the structures of the schema, view, and update, where possible, rather than the database extension. The motivation is to perform update interpretation in a syntactic manner, independent of the particular database state.

3.2.1. Reference and opacity

PIQUE generates mainly candidate updates that can be directly derived from the user's phrasing of the request. This limitation is justified by observing that most reasonable updates correspond to different readings of expressions in *referentially opaque* contexts.

A referentially opaque context is one in which two expressions that refer to the same real world concept cannot be interchanged in the context without changing the meaning of the utterance [Quine, 1961]. Natural language database updates often contain opaque contexts.

For example, consider that a particular individual, in a suitable database, may be referred to as "Dr. Smith", "the instructor of CS100", "the youngest assistant professor", or "the occupant of Room 424". While each of these expressions may identify the same database record (i.e., they have the same *extension*), they suggest different methods for locating that record (their *intensions* differ). In the context of a database query, where the goal is to unambiguously specify the response set, the method by which they are accessed does not normally affect the response. Updates, on the other hand, are often sensitive to the substitution of extensionally equivalent referring expressions. "Change the instructor of

CS100 to Dr. Jones" may not be equivalent to "Change the youngest assistant professor to Dr. Jones" or "Change Dr. Smith to Dr. Jones." Each of these may imply different updates to the underlying database.

For operating with an expression in an opaque context, therefore, we must consider the *sense* of the expression, in addition to its *referent* [Frege, 1952]. In a database system, this sense is embodied in the procedure used to evaluate the referring expression; the referent is the entity obtained via this evaluation. A request for a *change* to a referring expression is thus not specifically a request to perform a substitution on the referent of the expression, but rather a request to change the database so that the sense of the expression now has a new referent. That is, after the update, evaluating the same procedure should yield the new (requested) result.

Example 3.4:
Consider a database of ships, ports, and docks, where ships are associated with docks, and docks with ports. Assume that there is currently a ship named Totor in dock 12 in Naples (and no other ship in Naples), and consider the following update requests:

Change *Totor* to Pequod.

Change *the ship in dock 12* to Pequod.

Change *the ship in Naples* to Pequod.

The referring expressions (italicized) have the same referent in all three cases, but the senses differ. The expression "Totor" is resolved via a lookup in the *ships* relation; "the ship in dock 12" requires a join between the *ships* and *docks* relations; "the ship in Naples" requires a join between all three relations.

Consider the ways of performing each request, as indicated by the sense of the referring expression. The first version can be implemented only by making a direct substitution on the *ships* relation, corresponding to renaming the ship. The second admits this possibility, but also the possibility of moving a new ship into the dock (if there is already a ship named Pequod). The third allows the above two, plus the possibility of moving a different dock into Naples (if there is a dock somewhere else with Pequod in it).[2] Thus, the particular

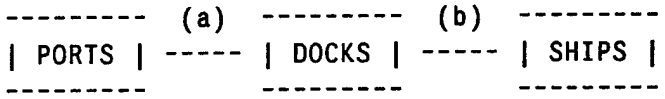
[2]This translation will later be ruled out for other reasons, but cannot be excluded on purely linguistic grounds.

referring expression selected by the user motivates a set of possible actions that may be appropriately taken, but does not directly indicate which is intended or preferred.

This characteristic of natural language updates suggests that the generation of candidate updates can be performed through *language driven inference* [Kaplan, 1979] without severely limiting the class of updates to be examined. "Language driven inference" is a style of natural language processing in which the inference process is driven (and hence limited) by the phrasing of the user's request.

In this instance, the candidate updates are generated by examining the referring expression present in the update request. This expression is evaluated by following an "access path" through the database structure. The candidate updates computed by the program consist of changing links or pointers along that path, or substituting values in the final record(s) identified.

For example, consider the structure of the "ships" database:



The candidate translations for the third request (changing "the ship in Naples") correspond to the following changes to the database:

(1) making a change to the *ships* file (i.e., renaming the ship);

(2) changing link (b) (moving a new ship into the dock);

(3) changing link (a) (moving a new dock into the port).

If the expression "the ship in dock 12" were used, only options 1 and 2 would be generated; similarly, if "Totor" were used, only option 1 would be generated.

3.2.2. Classes of views and updates

As discussed in Section 3.1, the views allowed by PIQUE are those corresponding to chain-structured conjunctive queries. The set of acceptable update requests is similarly restricted. The *qualification* of the update must be a conjunction of equi-selection terms. Dayal ([Dayal, 1978], [Dayal and Bernstein, 1982]) refers to these updates as *simple* updates.

As in other aspects of the system, we further restrict the set of updates considered to those corresponding to changes that can be specified naturally with a natural language interface. This is manifest in two restrictions:

(a) only deletion and replacement are considered, not insertion;

(b) for a replacement:

-the new value for an attribute must be a constant, not an indexed attribute;

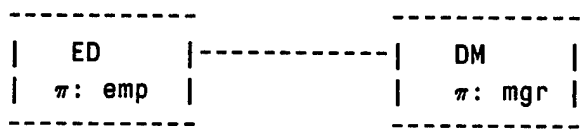
-the set of attributes to be updated must consist of either a single join attribute or a set of non-join attributes.

The latter restriction is imposed for reasons of simplicity in further processing; it can be justified by observing that updates violating this restriction are complex and would probably be expressed in natural language as multiple, conjoined requests. The "coverage" of PIQUE is discussed further in Chapter 5.

The emphasis in PIQUE is on the interpretation of *replacement* requests. Deletion is included only because it is a simpler case, and can be handled with similar machinery.

3.2.3. Query graph

A conjunctive query (and hence, a PIQUE view) can be represented graphically in a revised form of a *query graph* [Finkelstein, 1982], consisting of nodes (representing relations) and arcs (representing joins between relations). Projections and selections may be indicated by labels attached to the appropriate nodes. For example, query Q1 of Example 2.1, "List the employees and their managers", would be modelled:



The query graph is an analogue of the query expression and represents the *intension* of the query.

The algorithm for generation of candidate translations operates on the query graph corresponding to the view. Observe that the query graph corresponding to a PIQUE view (i.e. a chain-structured conjunctive query) will consist of a linear chain of joins. The generation of candidates is determined by the intension of the view and update and does

not depend on the current contents of the database, although the feasibility of use of certain candidates may be conditional upon finding favorable values in the database.

The view extension consists of a set of chains of tuples, each matching the structure presented by the query graph. The update selects a subset of these chains to be acted upon.

3.2.4. Terminology

The update request identifies certain tuples of the view and specifies changes to them (replacements or deletions). Call these view tuples the *marked tuples.*

Certain attributes of the view appear in the qualification of the update request. These are called the *qualifying* attributes. Similarly, attributes that appear in the target of a replacement request are called *target* attributes. Note that the two sets need not be distinct.

We introduce the following notation: for relation R_i, we will use $\mathbf{t}_i$ to represent a tuple variable whose range is R_i, and t_i, t_i', etc, to represent actual tuples of R_i.

Let V be a view definition, and v a tuple in the extension of V. Assume that the set of relations underlying V (i.e., that appear either in the *qualification* or *target* of V) is $R_1, \ldots, R_p$. Let $(t_1, \ldots, t_p)$ be a set of tuples, one from each of the underlying relations, such that $V(t_1, \ldots, t_p) = v$. Then we say that $(t_1, \ldots, t_p)$ is a *generator* of v. Each t_i, $i=1, \ldots, p$, is a *source-tuple* of v.

Let $v0$ be a set of view tuples. Let *generators*($v0$) be the set of generators of $v0$ (i.e., the union of the generators of all $v \in v0$). Then a *source* of $v0$ is defined as follows: for each $g \in$ *generators*($v0$), let $A(g)$ be some tuple of g; then, $s = \cup\{A(g) \mid g \in \textit{generators}(v0)\}$ is a *source* of $v0$.

Intuitively, the source includes at least one tuple from every generator. A source of a set of view tuples *supports* those tuples, so that if the source is deleted or changed, so are all the view tuples. This definition of *generator* is identical to that used by Dayal and Bernstein; the definition of *source* is similar to theirs.

3.2.5. Acceptable translations

An update translation should at least perform the change requested by the user, although it may have unrequested side effects. Specifically,

(a) if an update request is a *deletion*, the specified tuples must not be present in the new state (other tuples may be deleted, modified, or inserted);

(b) if the request is a *replacement*, the specified tuples must be modified in the requested way (they may be modified in other ways, and other tuples may be modified, deleted, or inserted).

In the terminology of Dayal and Bernstein, the only translations considered are those that *perform* the update (although our notion of performance is slightly broader than theirs).

The class of permissible translations is restricted in a number of ways, again, to ensure that the translations are "reasonable".

(a) The *type* of the translation (deletion or replacement) must match the type of the request. Sometimes a deletion, for example, can be effected by performing a particular replacement; such translations are disallowed. This restriction is also present in the work of Dayal and Bernstein.

(b) The translation must affect only tuples "involved" in the view (i.e. tuples that support view tuples selected for update). A view tuple corresponds, under the inverse of the view definition function, to a set of *source tuples* from the underlying database. Translations are allowed to affect only source tuples of the marked view tuples. This restriction eliminates a plethora of "unreasonable" translations and is present in the work of Keller [Keller, 1982], and in [Chamberlin, Gray, and Traiger, 1975] as the *rectangle rule.*

(c) A single conceptual change in the view must map into a single change to the database. If multiple view tuples are to be updated, the *generators* supporting them must all be updated in the same way. This ensures that the update request, which is conceptually a single operation on the view, is not translated into conceptually diverse actions on the database. Each translation must be expressible as a single DML operation on the database.

(d) The translation must be *minimal*, in that no proper subset of the changes specified in the translation will perform the update.

3.2.6. The generation algorithm

We present the actual method of generating translations in more detail. Deletion and replacement are treated separately.

3.2.6.1. Deletion

To effect deletion of a set of view tuples, it is necessary to delete a *source* of the tuples in question—i.e., to remove at least one tuple from each generator. We consider only translations that delete exactly one tuple from each generator. Translations that delete more tuples are non-minimal. (Though, for an example of a case where such a translation is reasonable, see [Keller, 1982]).

In accordance with restriction (c) of the previous subsection, the source tuples are all required to be from the same relation. Thus we seek to delete a *single-relation source* of the set of marked view tuples. This approach is used in the work of Dayal and Bernstein.

The actual generation algorithm considers deletion of relevant tuples (i.e., members of generators of marked tuples) from each of the relations which support the view. Formally: for each relation R_i underlying the view V, generate a translation of the form:

"Delete all the tuples of R_i that are source-tuples of the marked view tuples."

Thus, given the view presented previously:

```
{ x∈ED, y∈DM | y.dept=x.dept : x.emp, y.mgr }
```

and a request to delete the tuple(s) `(Brown, Jones)`, the translations generated are:

```
{ x∈ED, y∈DM | x.emp="Brown" ∧ y.mgr="Jones" ∧ y.dept=x.dept :
  delete x }

{ x∈ED, y∈DM | x.emp="Brown" ∧ y.mgr="Jones" ∧ y.dept=x.dept :
  delete y }
```

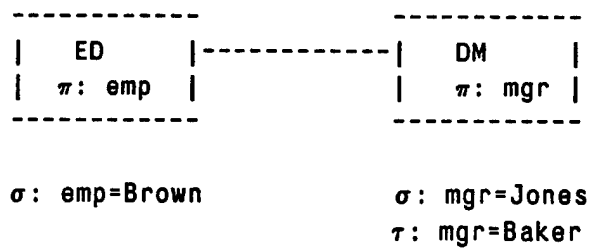
Translations generated in this way will always delete all the required view tuples, and will therefore successfully perform the update request. In addition, since exactly one tuple is deleted from each generator, the translations are *minimal*; no subsets of the deletions will

fulfill the request. As indicated, this algorithm finds all translations consisting of deletion to single-relation sources, and thus finds the complete set of acceptable translations.

3.2.6.2. Replacement

Replacement is considerably more complicated than deletion, for a number of reasons. The set of potential translations of a replacement request is larger and more diverse than for a deletion request. Some of these translations may be *contingent* upon finding a particular tuple or set of tuples in the database. Finally, the possibility of constraint violation is larger; for example, a replacement, unlike a deletion, may violate a functional dependency.

We will illustrate the generation with reference to Example 2.1, and then generalize. The request in the example was "Change Brown's manager from Jones to Baker", which can be represented in the query graph as follows:



That is, the update request specifies *emp* and *mgr* as qualifying attributes, and *mgr* as the target attribute.

One candidate translation consists of a change directly to the target attribute:

```
{ y∈DM, ∃x∈ED | x.emp="Brown" ∧ y.mgr="Jones" ∧ y.dept=x.dept :
  y.mgr←"Baker" }
```

There is also one translation consisting of a change to a link. This type of change will be called an *indirect* translation, and its generation can best be described with reference to the query graph. For Example 2.1, the only indirect change will be:

```
{ x∈ED, ∃y∈DM | x.emp="Brown" ∧ y.mgr="Jones" ∧ y.dept=x.dept :
  x.dept←"Mkting" }
```

which changes the link between the two relations.

Terminology: As the result of a translation which changes a join attribute (such as the second translation above), certain view attributes will be changed in an indirect manner, by virtue of being components of tuples that become part of generators. Call these the *affected* view attributes for the translation in question. The relations to which they belong are the *affected* relations.

In general, the indirect translations work by replacing each generator $(t_1, \ldots, t_{i-1}, t_i, t_j, \ldots, t_p)$ with a new generator $(t_1, \ldots, t_{i-1}, t_i^r, t_j', \ldots, t_p')$, by changing t_i to t_i^r. (Note that the t' tuples, which are incorporated into generators because of the change to t_i, are not necessarily all distinct from the t tuples; $t_j' \neq t_j$, however.) The affected relations will be those relations in the connected subgraph of the query graph consisting of the relation opposite the changed attribute, and all its descendants, i.e. $\{R_j, \ldots, R_p\}$. In the second translation for the example above, the only affected relation is DM, and the only affected view attribute is the target attribute itself, *mgr*.

In order for the translation to perform the update, the target attribute(s) must be included in the affected attributes. For example, if a translation for the above example consisted of a change to the department of Baker (i.e. modifying the tuple for Baker to make him the manager of Brown), the set of affected attributes, *emp*, would not include the target attribute. Thus, this translation could not successfully perform the request. Such a translation would leave Jones as a manager of Brown, and thus violate the definition of performance. In addition, the translation would violate the rectangle rule, by making a change to a tuple that did not support the marked view tuple.

We elaborate slightly on the definition of *performance* of a replacement request: In the new view tuple, the values of target attributes must have been modified in the requested way (thus, in the example above, the new value for *mgr* had to be "Baker"), and the values for qualifying attributes must not have changed (thus, the *emp* attribute had to remain "Brown", rather than being changed to, say, "Adams"). Other attributes may change; there were none of these in the example.

We now describe the candidate generation process for the general case. As with deletion, the change must be made to the set of generators of the marked tuples—i.e. the change is made to a *source* of the marked tuples. All generators must be changed in the same way.

Note that, observing the rectangle rule, the translation may not change tuples that are not members of a generator of a marked view tuple.

One candidate translation is to make the required change(s) directly to the target attribute(s). This translation always succeeds (except for possible constraint violation).

Other translations will consist of changes to join attributes. These translations will retarget links to incorporate new tuples in the appropriate generators. Since the set of affected attributes (which is determined by which end of the link is being changed) must contain the target attributes, only one end of the join may serve as a location for changes that perform the update. Note that if there are target attributes on both sides of the link in question (i.e., in both of the subgraphs that remain when this link is removed from the query graph), there will be no way to perform the update request by modifying this link.

More formally: for each join $\mathbf{t}_i.Y = \mathbf{t}_j.X$ in the view V, determine which end (if either) may be changed so as to include the target attributes in the set of affected attributes (say, $\mathbf{t}_i.Y$). For each tuple t_i of R_i that is a source tuple of a marked view tuple, change the chosen attribute t_i[Y] to a value such that a new generator is formed, which supports a new view tuple with the properties that:

(1) all target attributes have their appropriate (new) value;

(2) all qualifying attributes have their appropriate (old) value (except those that also serve as target attributes).

In many cases the state of the database will be such that appropriate tuples do not exist, and the translation cannot be performed by changing the link in question; or, there may be multiple appropriate values. This is the *contingency* problem of indirect translations.

a. Contingent translations

While the set of acceptable translations of a deletion request is independent of the state of the database, some of the translations of replacement requests are *contingent* upon the presence of certain tuples in the database. Thus, in Example 2.1, the translation of moving Brown to another department depends upon finding a department that is managed by Baker. For reasons of clarity and uniformity, it is important that the *structural* (intensional) aspects of the translation be separated from those aspects that are dependent on the current database extension. Thus, in the example, the translation in question means

roughly "Move Brown to a department that is currently managed by Baker." The process of "instantiating" this translation—finding out whether such a department exists, and if it is unique—is separate.

A translation such as this cannot be expressed directly in the data manipulation language. Thus we introduce a new construct to permit expressions of this sort. The indirect translation for the example would appear as:

```
{ x∈ED, ∃y∈DM |
  x.emp="Brown" ∧ y.mgr="Jones" ∧ y.dept=x.dept :
  x.dept←y'.dept
       {Each y'∈dm | y'.mgr="Baker"}}
```

The "Each" indicates that we consider separately each department meeting the restrictions. This is discussed below under instantiation.

The need to extend the DML for expressing updates has arisen in other cases. Keller, in discussing insertion into views formed by joins, points out that some of the necessary tuples might already be present in the database [Keller, 1982]. Thus, the number of tuples to be inserted may vary, depending on the state of the database, and the translation cannot be expressed as a single request in the DML. Keller suggests minor extensions to the intermediate language to remedy this problem. Similarly, Dayal and Bernstein discuss the necessity of checking that the translation satisfies integrity constraints [Dayal and Bernstein, 1982]. Their method is to encode run-time integrity checks into the translation; this approach in general requires that the translation be written in a general-purpose programming language, rather than a data-manipulation language. In both of these cases, the motivations for extending the access language are different from ours, but the methods are similar: effectively, the access language is augmented with a form of conditional.

b. Instantiation

Once the intensional form of a contingent translation has been generated, it must be instantiated by retrieving the appropriate values from the database. In the current implementation of PIQUE, this process is carried out as part of the generation phase. For efficiency, it may be desirable to defer this until after ranking.

Referring again to the example, the instantiation consists of retrieving the set of departments that are already managed by Baker:

```
{ y'∈DM | y'.mgr = "Baker" : y'.dept }
```

If this set is empty, no translation of this form is possible. If the set has multiple elements, each of them will be used in a separate candidate translation.

In certain cases, the integrity constraints expressed in the database schema may be sufficiently strong to guarantee that the set of elements is empty, without the necessity of instantiation. Discussion of this topic is deferred to Subsection 3.3.4, "Integrity Constraints".

c. Completeness

As indicated in the restrictions of Subsection 3.2.5, translations must consist of changes to single relations, except where noted. Thus, except for the *direct* translation, a translation must consist of a change to a single link. By the properties of affected attributes, only a single translation is possible at each link. The algorithm above finds all such changes that *perform* the request, according to the definition of performance. Thus, the algorithm finds all acceptable translations.

3.3. Ranking Candidate Translations

The generation phase produces a set of candidate translations, all of which perform the requested update. The next operation is to choose among them. In PIQUE, the basis for the choice is provided by a set of heuristics, which reflect the appropriateness of a particular translation as a response to the update request. The heuristics impose an ordering on the candidates; in general only a partial ordering may be obtained, since the heuristics may be insufficient to distinguish among translations.

It is important to separate the specific details—the heuristics used by PIQUE—from the general method—ranking candidate translations. The heuristics used in PIQUE are motivated by considerations of availability and portability: they do not require provision of additional information beyond that normally found in database applications. In other situations, different heuristics may be desirable.

PIQUE uses heuristics based on four conceptually distinct sources of information: accuracy considerations, pragmatic information, semantic information, and integrity constraints. These are discussed in turn.

3.3.1. Accuracy considerations

The candidate translations are those that *perform* the user's request, while meeting the restrictions of appropriateness. Ideally, the translations should *exactly perform* the request, with no unrequested changes to the user's view. In such a case, the commutativity illustrated in the diagram of Section 2.2 holds. In general, however, translations may introduce side effects, and a given request may have no associated translation which exactly performs it.

- A heuristic used by PIQUE is to assign highest ranking to the translations with minimal side effects.

This heuristic would apply in Example 2.1. Of the two translations generated, candidate (b)—moving Brown to a different department—would have no side effects (and would *exactly perform* the requested update), whereas candidate (a)—changing the manager of Brown's department—would have the side effect of changing the manager of two other employees. Candidate (b) would therefore be preferred.

A more compelling basis than actual side effects is the consideration of *potential* side effects, as determined from the underlying schema. The actual side effects result from a particular state of the database; the potential ones take into consideration the types of side effects that can arise in all valid database extensions. These can be derived using information about the cardinality of relationships, as specified in the database schema; they do not require reference to the database extension.

In the structural model of a database [Wiederhold and El-Masri, 1980], which is used to encode the PIQUE database schema, such cardinality information is represented in *connections.* Specifically, connections record restrictions on *cardinality*—maximum number of corresponding entities (e.g. an employee may work for no more than one department)—and *dependency*—minimum number of corresponding entities (e.g. each department must have at least one employee). For the PIQUE schema, only one type of connection is used: the *reference* connection. A reference connection represents a join between a key of one relation, and key or non-key attribute(s) of another relation, in which every tuple of the second relation participates. The joins represented by reference connections are thus *extension joins* [Honeyman, 1980]. (This type of connection arises

from normalization to second normal form.) The semantics of the reference connection combines a functional dependency, which is implied by the structure of the connection, with a reference constraint.

In Example 2.1, there is a reference connection from ED to DM, specifically, from the dependent part of ED to the key of DM. Each ED tuple is associated with exactly one DM tuple. Note that relations in the structural model are in Boyce-Codd normal form: the only functional dependencies are those implied by keys of relations.

Consider again the potential side effects for Example 2.1. Irrespective of the particular state of the database, translation (b) cannot have any side effects. The cardinality of the employee-department relationship (N:1) ensures that each employee will be associated with at most one department, so a change to an employee can alter at most one tuple from the user's view. No such restriction holds on departments (i.e. a department may be associated with several employees), so changing the manager of a department may result in changes to several view tuples. For the particular database extension used in the example, two extra tuples are affected. Potential side effects are more meaningful than actual ones as a basis for comparison of translations, since they result from inherent characteristics of the domain, rather than fortuitous aspects of the current database extension.

The following subsection presents some algorithms used by PIQUE for computing the potential side effects of a candidate translation, based on the update request and the cardinalities of the connections in the view. This extends the work of Dayal and Bernstein ([Dayal, 1978], [Dayal and Bernstein, 1982]). We first introduce a graphical representation for the view and update, and present and prove a necessary lemma. The lemma is then used to establish structural conditions for the presence and absence of (different types of) side effects for update translations. Deletion and replacement, are considered separately, since they have different properties. Finally, we consider the problems of comparing the side effects of different translations and prove the *optimality* of certain translations in terms of minimal side effects.

3.3.1.1. View-dependency graphs

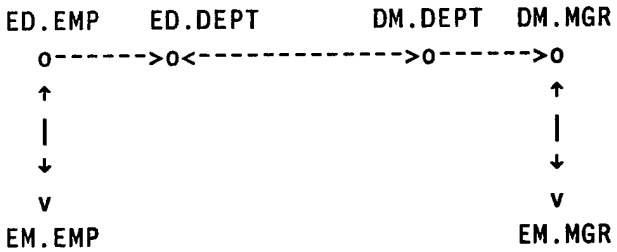
The calculation of potential side effects is performed using a graphical representation for

the user view and update. We first introduce a general notation, then specialize it for views encoded in the structural model.

Following the work of Dayal and Bernstein, we define a graph: given a view, the *view-dependency graph* is a directed graph with nodes and arcs defined as follows:

For every relation R_i underlying the view, for every attribute Y of R_i, there is an **o**-node labelled $R_i.Y$. For every view attribute Y which is a projection of $\mathbf{t}_i.Y$, there is a **v**-node labelled $V.Y$ and arcs $(R_i.Y, V.Y)$ and $(V.Y, R_i.Y)$. For every equi-selection clause $\mathbf{t}_i.Y$="c" in the qualification of the view, there is a **v**-node labelled "c" (a constant node) and arcs $(R_i.Y$, "c") and ("c", $R_i.Y)$. For every join $\mathbf{t}_i.Y = \mathbf{t}_j.X$, there are arcs $(R_i.Y, R_j.X)$ and $(R_j.X, R_i.Y)$. Finally, for every functional dependency f: $X \underset{Ri}{\rightarrow} Y$, there is an arc $(R_i.X, R_i.Y)$. (We assume that both X and Y are singleton sets.)

The view-dependency graph for Example 2.1 is:



We define *paths* in the graph in the obvious way. There is a path from a node X to a node Y (written $X \underset{v}{\rightarrow} Y$) if:

(a) Y and X are the same node; or,

(b) there is an arc (X,Y); or,

(c) there is an intermediate node Y_0 such that $X \underset{v}{\rightarrow} Y_0$ and $Y_0 \underset{v}{\rightarrow} Y$.

Paths can be defined on sets of nodes. There is a path from a set of nodes L to a node Y if there is a path from a subset of L to Y. There is a path from a set of nodes L to a set of nodes M if there is a path from L to each node of M.

We first introduce a lemma, drawing from the work of Dayal and Bernstein:

Lemma 1: Let M be the set of attributes of relations underlying the view, L be a subset of M, Y a set of view attributes, and E the set of attributes appearing in equi-selections in the view definition. If

$(L \cup E) \not\xrightarrow[V]{} Y$

then there is a semantically consistent state in which there exist generators g, g', of view tuples v, v', such that $g[L] = g'[L]$, $v[Y] \neq v'[Y]$.

Proof: A similar lemma is proven by Dayal [Dayal, 1978]. We repeat the proof here. Assume, without loss of generality, that Y is a singleton, and that it is the projection of $R_k.A$. Let the set of attributes of relation R_k be M_k.

First, suppose that only the relation R_k underlies the view. Then,

Since $L \not\xrightarrow[V]{} Y$, $\exists\, N \subset M_k$, $B \in M_k$ such that $N \rightarrow B$ and the view definition implies $\mathbf{t}_k.L = \mathbf{t}_k.N \wedge \mathbf{t}_k.A = \mathbf{t}_k.B$.

Since $E \not\xrightarrow[V]{} Y$, there is no selection in the view definition V, of the form $\mathbf{t}_k.A = \text{"c"}$.

Hence we can find tuples t_k, $t_k' \in R_k$ that are generators of view-tuples and are such that $R_k = \{t_k, t_k'\}$ is semantically consistent, $t_k[L] = t_k'[L]$, and $t_k[A] = y \neq y' = t_k'[A]$.

Next, suppose that relations $(R_1, \ldots R_p)$ underlie the view. Define a new relation scheme $R[M]$, where M is the co-product of the M_i for the relations R_i, such that the extension R is the product of the extensions R_i, $i = 1, \ldots, p$. The only inter-relational constraints are the reference constraints associated with the reference connections; these do not affect the presence or absence of generators. The functional dependencies are only intra-relational; it follows that $\exists f: Z \xrightarrow[Ri]{} W$ iff $\exists f': R_i.Z \xrightarrow[R]{} R_i.W$. Hence if we draw a view-dependency graph on R instead of on the R_i, then the new graph will be isomorphic (up to relabelling of nodes) to the original. Hence, by our arguments for the previous case (where only R_k was involved), we can find tuples t, $t' \in R$, such that t, t' are generators of view tuples and $R^0 = \{t, t'\}$ is a semantically consistent extension and $t[L] = t'[L]$ and $t[Y] \neq t'[Y]$.

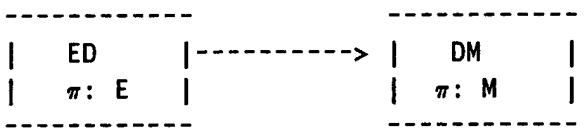
But R^0 is not a valid product of the $R^0[M_i]$. We show that augmenting R^0 to R, the product of the $R^0[M_i]$, does not violate semantic consistency. For, suppose that R does violate some functional dependency $R_i.Z \xrightarrow[R]{} R_i.W$:

$$\Rightarrow \exists\, u, u' \in R\,(u[R_i.Z] = u'[R_i.Z] \wedge u[R_i.W] \neq u'[R_i.W])$$
$$\Rightarrow \exists\, u_i, u_i' \in R_i = R[M_i]\ (u_i[Z] = u_i'[Z] \wedge u_i[W] \neq u_i'[W])$$
$$\Rightarrow \exists\, s, s' \in R^0\,(s[R_i.Z] = s'[R_i.Z] \wedge s[R_i.W] \neq s'[R_i.W])$$

which contradicts the hypothesis that R^0 is semantically consistent. Hence, we can augment R^0 to R without violating semantic consistency. Now, let $t_i = t[R_i.M_i]$,

$t_i' = t'[R_i.M_i]$ for all i. For all i, let $R_i = \{t_i, t_i'\}$. Thus we have constructed a semantically consistent extension with the desired properties. This completes the proof of the lemma. ∎

The formulation of graphs and paths can be specialized to the views used in PIQUE, in which all joins represent reference connections, and all relations are in Boyce-Codd normal form. Consider an extension of the query graph formalism of Section 3.2.3, with nodes representing relations, and arcs representing joins, labelled with arrows to show the direction of the associated reference connection. In this new form, the schema of Example 2.1 looks like:



Paths in the graph are defined by the arrows labelling the arcs. We use this graph to define paths between *attributes*, parallel to the development in the more general graph. There is a path from an attribute X (of relation R_i) to an attribute Y (also stated as: Y is *accessible* from X), written $X \xrightarrow[v]{} Y$, if either:

(a) X and Y are the same attribute; or,

(b) X is a key of relation R_i, and either Y is an attribute of R_i, or there is a path from R_i to the relation containing Y (say, R_j) following the arcs of the graph; or,

(c) the view definition contains a join $\mathbf{t}_i.X = \mathbf{t}_k.Z$, and there is a path from $R_k.Z$ to Y, of form (a) or (b).

Note that paths correspond to functional dependencies. An attribute functionally determines another if and only if there is a path from the first to the second. Also, we say that there is a path from a *relation* R_i to an attribute Y if there is a path from the key of R_i to Y. Similarly, there is a path from Y to R_i if there is a path from Y to the key of R_i. Thus, we speak of paths (and accessibility) between relations, as well as between attributes.

3.3.1.2. Calculation of potential side effects

The extended query graph is used to determine potential side effects. As observed by Dayal and Bernstein, an update translation that changes a single relation cannot have any side effects in the view if the relation changed is a *functionally determining source* of the

view. Graphically, this condition holds if paths exist from the relation in question to all the relations of the view, using the arrows representing the reference connection between relations—i.e., if the changed relation is a *root* of the graph. This is because each tuple in such a relation is only in the set of generators for one view tuple.

Consider again Example 2.1. From ED, there are paths to all view attributes, thus changes can be performed to ED without side effects. The same is not true of DM: the *emp* attribute is not accessible from the DM relation, and therefore changes to DM may have side effects.

This work on discovery of translations that have no side effects can be extended to encompass changes that have potential side effects. For example, in the employee-manager view, an attempt to delete a single tuple by performing the deletion on the DM relation may have the side effect of deleting other (*emp, mgr*) tuples which have the same manager. The query graph can be used to calculate potential side effects.

If the view contains *selections* besides the joins and projections, the calculation of side effects is slightly more complicated. The selections may further restrict the view intension, to guarantee absence of side effects in a larger class of situations. Thus, if the view definition for the above example had been, "List the employees managed by Jones.", neither of the translations would have side effects. In general, side effects in the view are avoided if all of the attributes that appear in the view are accessible, either from the changed relation, or from the attributes appearing in equi-selections in the view definition. (Note that this condition applies to the example just described.) In what follows, we will refer to these attributes as *equi-selected* attributes, and call the set of these attributes E. The calculation of paths from equi-selected attributes is the same as for other attributes. For clarity, most of our examples will involve only joins and projections.

We now discuss the calculation of potential side effects for candidate translations.

Terminology: In any translation, all updates are performed to a single relation. Call this the *changed relation*, R_C.

Notation: In what follows, we will often want to consider whether every member of a given set of attributes (say, M) is accessible either from R_C or from E. Instead of writing

$\forall\ Y \in M\,(R_C \overset{\rightarrow}{\underset{V}{}} Y \vee E \overset{\rightarrow}{\underset{V}{}} Y)$ we write $(R_C \cup E) \overset{\rightarrow}{\underset{V}{}} M$, with the understanding that the latter form involves some mixing of types.

a. Deletion

We establish conditions for the presence and absence of side effects for deletion requests.

Theorem 1. Let V be the set of view attributes. If $(R_C \cup E) \overset{\rightarrow}{\underset{V}{}} V$, then there will be no side effects, for any deletion.

Proof: This is also shown in [Dayal and Bernstein, 1982] and [Keller, 1982]. Essentially, each tuple of the changed relation supports exactly one view tuple, so the update will be performed exactly. ∎

Example 3.5:

```
ED ---> DM
{ emp="Brown" | delete }
```

There are paths from the ED relation to all the relations of the view, so removal of tuples from ED cannot have any side effects. Removal from DM may cause side effects—specifically, deletion of other employees who have this department and manager.

Dayal generalizes the condition concerning lack of side effects, in a stronger result, by considering presence or absence of side effects *for particular update requests.* To ensure exactness, there need to be paths from the relation in question, not to all of the attributes in the view, but only to those that appear as *qualifying attributes* in the update request.

Theorem 2. Let Q be the set of attributes that appear in the qualification of the update request. Iff $(R_C \cup E) \overset{\rightarrow}{\underset{V}{}} Q$, there will be no side effects.

Proof: This is also proven by Dayal and Bernstein.

If: $(R_C \cup E)$ functionally determine the values of Q, which determines whether tuples are marked or not. Thus, each tuple of R_C supports either only marked tuples or only unmarked tuples, and exactly the marked tuples will be deleted.

Only if: We show this by contradiction. Consider a generator of a marked tuple v, and the tuple t of R_C that is a member of that generator. If $(R_C \cup E) \not\overset{\rightarrow}{\underset{V}{}} Q$, by Lemma

1 t may also support a view tuple v' which differs in the value of one the qualifying attributes Q (i.e., is not a marked tuple). The tuple v' will also be deleted, which is a side effect. ∎

Example 3.6:
Consider the above view, and the request

```
{ mgr="Baker" | delete }
```

There is only one qualifying attribute, *mgr*, which is accessible from both ED and DM, thus deletion can be done from ED, as before, or from DM. Removing all the DM tuples with manager Baker will have no side effects.

Theorem 3: Side effects, if any, will consist of additional deletions.

Proof: The view definitions used here are *additive* [Finkelstein, 1982]; formally, $V(A \cup B) = V(A) \cup V(B)$. It follows from the property of additiveness that deletion from the base relations can cause only deletions in the view. ∎

Corollary 3.1: Side effects are calculated as follows:
The tuples deleted will all have correct (i.e., marked) values for attributes that are accessible from the changed relation (or from equi-selected attributes). They may have incorrect values for qualifying attributes that are not so determined.

Proof: This follows from Theorem 2 and Theorem 3. As shown in Theorem 2, deleted tuples may support view tuples that are correct except for the value of the qualifying attributes. These tuples will be deleted. ∎

Example 3.7:
Consider a more complicated view, containing relations for administrators and locations, with departments and administrators related via their location:

```
ED  --->  DL  --->  L  <---  AL
```

Assume the request is:

```
{ emp="Brown" ∧ adm="Forsythe" | delete }
```

Consider the effects of deletion from each of the relations:

ED: will delete other tuples with the correct employee, department, and location, but wrong administrator (i.e., different administrators for this employee);

DL: will delete other tuples with the correct department and location, but the wrong employee or administrator (i.e., other administrators for this employee, other employees for this administrator, or tuples with both wrong);

L: will delete all tuples with this location (i.e., all employees who work in departments with this location, all administrators who have this location);

AL: will delete tuples with the appropriate administrator and location, but the wrong employee and department (i.e., other employees for this administrator);

To evaluate possible side effects for deletion from ED, we see that ED $\underset{v}{\rightarrow}$ {dept, loc}; but the connection from L to AL points the "wrong" way. The tuples deleted will all have the correct value for "employee", but may have an incorrect value for "administrator"—in more intuitive terms, other administrators for the same employee may be deleted. This can be used if it is necessary to warn the user of the potential side effects.

The side effects for deletion from DL are a superset of those for deleting from ED. Similarly, deletion from L is worse than deletion from any other relation. Note that the effects of deletion from ED and AL cannot be compared; neither is a superset of the other. (Note also that in the structural model, deletions from DL and L would be blocked by the semantics of the reference constraint of the reference connection; they are included here for completeness in the treatment of side effects.)

b. Replacement

The derivation of side effects for replacement proceeds as for deletion, using the query graph and update request, except that the range of possible side effects is much larger. We identify three types of side effects, and the conditions under which they arise. As with deletion, the first theorem concerns a class of views to which all updates can be performed safely.

Theorem 4. Let V be the set of view attributes. If $(R_C \cup E) \underset{v}{\rightarrow} V$, exactly the marked tuples will be updated, for all updates.

Proof: Analogous to the proof of Theorem 1. ∎

Again, this can be strengthened, by considering the individual update request.

Theorem 5. Let Q be the set of qualifying attributes for the update request. Iff $(R_C \cup E) \xrightarrow[V]{} Q$, exactly the marked tuples will be updated.

Proof: Analogous to the proof of Theorem 2. ∎

Example 3.8:

```
ED --->  DM
{ emp="Brown" ∧ mgr="Jones" | mgr←"Baker" }
```

The change can be made to ED or DM. The change to ED cannot have any side effects; however, it is contingent on the presence of a DM tuple with manager "Baker". The change to DM can always be made, but may cause side effects of changes to the managers of other employees. This type of side effect—making the requested change to extra view tuples, will be called a *Type 1* side effect.

Example 3.9:
Consider the above view, and the request:

```
{ mgr="Jones" | mgr←"Baker" }
```

The update may be made exactly to either relation, since the qualifying attributes are functionally determined by both.

The next two theorems introduce two additional types of side effects. These arise only with replacement (as opposed to deletion) requests.

First, note that if the translation consists only of changes to *non-join* attributes, view tuples will be changed only in the requested way, and no insertions or deletions will be caused. This follows from the properties of the candidate generation algorithm, which was discussed in Section 3.2.

The following theorems apply to translations in which a *join* attribute is changed.

Theorem 6: For a change to a join attribute:
Let A be the set of *affected* view attributes for the translation, as defined in Section 3.2.6. The only change to the marked tuples will be the requested one, iff all attributes in A are:

(a) mentioned in the update request (in either qualification or target), or

(b) accessible from the affected qualifying attributes of the update request that do not appear in the target, or

(c) accessible from affected attributes that appear in equi-selections in the view definition.

If the above condition does not hold, attributes other than the target attributes may change. This kind of side effect—making an extra change to the appropriate tuples—will be called a *Type 2* side effect.

Proof:

If: If affected attributes are mentioned in the qualification or target, the candidate generation algorithm ensures that they have the correct value after the update. Similarly, if they are accessible from affected qualifying attributes that do not appear in the target or affected attributes that appear in equi-selections—i.e., attributes whose values do not change during the update—their values will not change.

Only if: Let Q be the set of qualification attributes of the update request, Q' be those members of Q that are affected attributes. Define E, E' similarly, for attributes that appear in equi-selections in the view definition. If the above condition does not hold, there are affected attribute(s) that are neither qualification nor target attributes, nor accessible from Q' or E'; call this set of attributes U.

By Lemma 1, since $(Q' \cup E') \not\stackrel{*}{\rightarrow} U$, there are semantically consistent states in which the new generators have the appropriate values on the attributes in Q' and E' (and thus satisfy the candidate generation conditions), but have different values on U. The new values for the attributes in U will be side effects of the update.

Note that there may be paths to U from attributes in $(Q - Q')$ or $(E - E')$; since the link from these attributes to U is changed during the update, these dependencies do not affect the proof. ∎

Example 3.10 (Type 2 side effects):

```
ED --> DM; π: emp, dept, mgr
{  emp="Brown" | dept← "Mkting" }
```

The qualifying attributes are {emp}, target attributes are {dept}.

If Brown's department is changed to Marketing, the affected attributes will be {dept, mgr}. As a side effect of this update Brown's manager will probably be changed. Note that we allow such translations, because they may represent the user's intentions—i.e., it is possible that the user expected the manager to change, but did not mention this in his request.

Theorem 7. For a change to a join attribute:
Let A be the set of affected view attributes for the translation. Iff $(R_C \cup E) \underset{v}{\rightarrow} A$, the *number* of view tuples will not change; the only effects in the view will be replacement.

If this condition does not hold, tuples may be inserted and deleted. We refer to these changes as *Type 3* side effects.

Proof: Again, consider the change as replacing generators $(t_1, \ldots, t_{i-1}, t_i, t_j, \ldots, t_p)$ with new generators $(t_1, \ldots, t_{i-1}, t_i^r, t_j', \ldots, t_p')$.

If: If the condition holds, each tuple of the changed relation R_i links to exactly one chain of tuples in $(R_j, \ldots, R_p)$, both before and after the update. Thus the number of tuples does not change.

Only if: If the condition does not hold, there is an affected view attribute that is not accessible from $(R_i \cup E)$. By Lemma 1, there is a semantically consistent state in which there is a generator $g''=(t_1, \ldots, t_i^r, t_j'', \ldots, t_p'')$ (t'' not necessarily all distinct from t') of a view tuple v'', which is different from v', the requested result of the update. The tuple v'' is inserted into the view as a result of the update.

It remains to show that this state is consistent with the initial state—i.e., that the t' and t'' tuples are consistent with the t tuples. The consistency of t' with t follows from the generation algorithm, which requires that the states be consistent. The consistency of the t'' follows similarly: for each relation R_n, $(n=j, \ldots, p)$, either $(R_C \cup E) \underset{v}{\rightarrow} R_n$, in which case $t_n''=t_n'$, or $(R_C \cup E) \not\underset{v}{\rightarrow} R_n$, in which case the t_n''

can be chosen to be consistent with t_n. Thus, the t'' are also consistent with t, and the total state is consistent. ∎

Example 3.11 (Type 3 side effects): Consider the view:

```
DL ---> L  <--- AL
```

and the request,

```
{ dept="advert" ∧ adm="Lafleur" | adm←"Erman" }
```

If the change is made to DL (i.e., changing the location of the appropriate department), extra (dept, adm) tuples may be inserted or deleted.

Finally, we consider a complex example in detail:

Example 3.12:

```
ED  --->  DL ---> L  <--- AL; π: emp, adm
{ emp="Brown" ∧ adm="LaFleur" | adm←"Erman" }
```

No relation functionally determines all the qualifying attributes, so all changes will involve potential side effects. For each translation, we show the side effects that are indicated by the theorems, and verify this by calculation.

$E = \emptyset$

$Q = \{\text{emp, adm}\}$

$T = \{\text{adm}\}$

Consider the effects of performing the update to each relation:

ED: Translation will be:

```
{ x∈ED, ∃y∈DL z∈L w∈AL |
  x.emp="Brown"∧y.dept=x.dept∧z.loc=y.loc∧w.loc=z.loc
     ∧ w.adm="LaFleur" :
  x.dept←y'.dept
        {Each y'∈DL ∃z'∈L w'∈AL |
              z'loc=y'.loc∧w'.loc=z'.loc∧w'.adm="Erman"} }
```

R_C = ED

$A = \{\text{adm}\}$

$(R_C \cup E) \xrightarrow[V]{} \{\text{emp, dept, loc}\}$;

$(R_C \cup E) \not\xrightarrow[C]{} Q \Rightarrow$ Type 1;

$(R_C \cup E) \not\xrightarrow{}_{V} A \Rightarrow$ Type 3.

Side effects: may insert/delete tuples for other administrators of this employee; specifically, all the administrators who have the same location as the former department will be deleted, and other administrators having the location of the new department will be inserted (Types 1 and 3).

DL: The translation will be:

```
{ y∈DL, ∃x∈ED z∈L w∈AL |
  x.emp="Brown"∧y.dept=x.dept∧z.loc=y.loc∧w.loc=z.loc
     ∧ w.adm="LaFleur" :
  y.loc←z'.loc
        {Each z'∈DL ∃w'∈AL | w'loc=z'.loc∧w'.adm="Erman"} }
```

R_C = DL

$A = \{adm\}$

$(R_C \cup E) \xrightarrow{}_{V} \{dept, loc\};$

$(R_C \cup E) \not\xrightarrow{}_{V} Q \Rightarrow$ Type 1;

$(R_C \cup E) \not\xrightarrow{}_{V} A \Rightarrow$ Type 3.

Side effects: This will have all the side effects on the employee "Brown" that are associated with the change to ED; it may also perform the same administrator changes to all other employees in Brown's department (Types 1 and 3).

DL: The change cannot be performed to this relation.

AL: The translation will be:

```
{ w∈AL ∃x∈ED y∈DL z∈L |
  x.emp="Brown"∧y.dept=x.dept∧z.loc=y.loc∧w.loc=z.loc
     ∧ w.adm="LaFleur" :
  w.adm←"Erman" }
```

R_C = AL

$A = \emptyset$

$(R_C \cup E) \xrightarrow{}_{V} \{adm, loc\};$

$(R_C \cup E) \not\xrightarrow{}_{V} Q \Rightarrow$ Type 1;

$(R_C \cup E) \xrightarrow{}_{V} A.$

Side effects: may change the manager of many other employees, specifically those who work in the same location as Brown's department (Type 1).

Note that the types of side effects are independent of each other, and may occur in combinations.

The actual heuristic used by PIQUE for ranking translations of replacement requests is:

- Translations with no side effects are preferable to translations with Type 1 or 2 side effects, which are preferable to translations with Type 3 side effects.

3.3.1.3. Comparing translations

The previous section discussed the presence and absence of side effects. A translation having no side effects of a given type is obviously preferable to a translation that has some, but in the general case such a translation may not exist. In such situations side effects must be compared to determine which is larger. This section discusses comparison of side effects for different translations and presents some theorems about optimality. As before, deletion and replacement are considered separately.

Terminology: Given two translations, operating on different relations, if the potential side effects of one are a subset (not necessarily a proper subset) of the other, we say that the relation of the former *subsumes* the relation of the latter. If the inclusion is proper, we say that the former relation *dominates* the latter. The import of subsumption is that a subsumed (or dominated) translation may be eliminated from further consideration.

a. Deletion

Theorem 8: Consider a connected subgraph of the query graph that is functionally determined, in the sense that there is one relation (call it the *root*) from which there are paths to all other relations in the subgraph. For such a subgraph, deletion from the root will have side effects that are a subset of those associated with deletion from elsewhere in the subgraph. The root is said to *subsume* the other relations in the subgraph.

Proof: The set of attributes accessible from the root of the subgraph will be a superset of the attributes accessible from any other relation in the subgraph. Since side effects are caused by changes to attributes that are *not* accessible, deletion from the root will be at least as accurate as deletion from any other relation in the subgraph. ∎

Consider again Example 3.7. There are two maximal functionally determined subgraphs:

```
ED ---> DL ---> L

L <--- AL
```

By Theorem 8, ED subsumes DL and L. AL subsumes L.

Thus we have shown that a deletion performed to a root is at least as accurate as one performed to any of the functionally determined non-roots. To compare different roots for accuracy, we must consider their respective positions in the query graph.

Terminology: Consider again the query graph formalism. For relations R_i, R_j, and R_k, if the graph is structured such that removal of the node for R_k results in separate subgraphs containing R_i and R_j, we say that R_k *separates* R_i and R_j. Pictorially, R_k separates R_i and R_j if it lies between them. Note that separation is independent of the directionality of paths that may exist between the nodes.

In addition, we extend the definition of *root* to incorporate the effects of equi-selections. A relation R_i is a root iff:

(a) there is no reference connection to it; and

(b) its key does not appear in an equi-selection in the view definition.

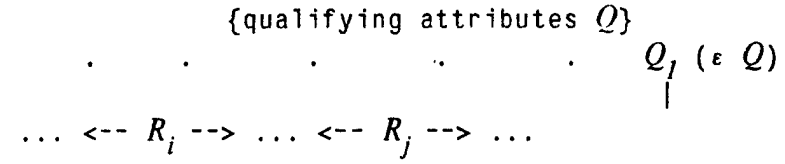
We introduce a lemma that enables comparison of roots, based on their position with respect to the qualifying attributes of the request.

Lemma 2: Let Q_I be one of the qualifying attributes of the deletion request, and let R_i and R_j be relations which appear as roots. If R_j separates R_i and the relation containing Q_I (or R_j contains Q_I) then R_i does not subsume R_j.

Proof: This follows from the fact that $R_i \not\Rightarrow R_j$, which follows in turn from the definition of root. Certain tuples of R_j will support view tuples with the marked values for the attribute Q_I (independent of other restrictions in the update request).

Consider deletion from R_i. The R_i tuples deleted will meet all restrictions, including the one on supporting view tuples with the marked value for Q_I. Thus, they will all join with tuples in the subset of R_j identified in the previous paragraph. But, since $R_i \not\Rightarrow R_j$, by Lemma 1 they may also join with R_j tuples *outside* that set. The extra view tuples supported by these R_i tuples will be deleted if the update is performed to R_i, but not if it is performed to R_j. Thus R_i does not subsume R_j. ∎

The situation described by the theorem can be represented pictorially with a simplified view-dependency graph:



The lower row represents the chain of joins that supports the view. R_i and R_j represent the roots of their respective functionally determined subgraphs; the "..." indicate unspecified sequences of joins. The upper row represents the view level, with the position of the qualifying attribute Q_l indicating its position with respect to the underlying chain of joins.

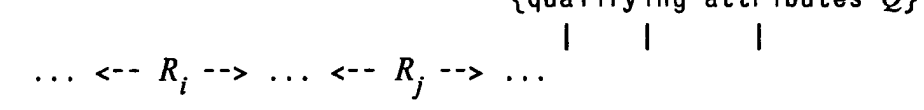
The lemma shows that, in this situation, R_i does not subsume R_j.

We use this lemma to prove two results about the side effects engendered in performing updates to different roots. The first identifies a class of requests in which the translations may be compared, and indicates which translation is optimal. The second proves that for a different class of requests, roots cannot be compared.

Theorem 9. Let Q be the qualifying attributes of the deletion request, and R_i, R_j be root relations. If, for each attribute Q_l of Q, R_j either contains Q_l, or separates R_i from the relation containing Q_l, then R_j dominates R_i.

Proof. Consider separately deletion from R_j and R_i. If the deletion is performed from R_j, the tuples deleted will be those supporting view tuples with the marked values for attributes in Q. If the deletion is performed from R_i, the tuples deleted from R_i will be exactly those that join with tuples in the subset of R_j identified above—i.e., those R_i tuples that are members of generators whose R_j tuple belongs to the specified subset. Thus, any view tuple which would be removed if the deletion is performed to R_j will also be removed if the deletion is performed from R_i, and all the side effects for R_j occur with R_i. As shown in the proof of Lemma 2, additional potential side effects will occur with deletion from R_i. Thus, R_j dominates R_i. ∎

The situation described by the above theorem is the following:

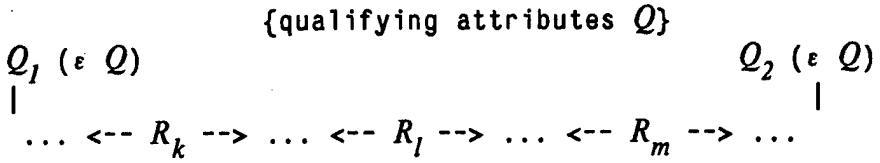


R_j dominates R_i.

Theorem 10. Let Q_1, Q_2 be two qualifying attributes for the deletion request, not from the same underlying relation. All roots that separate the relations containing Q_1 and Q_2 (or that themselves contain Q_1 or Q_2) will be *incommensurable*, in the sense that none will subsume any of the others.

Proof. We use Lemma 2 on each pair of roots. In each case, Lemma 2, applied to the roots and the qualifying attributes, shows that neither root subsumes the other. ∎

The situation described by the above theorem is the following:



None of R_k, R_l, R_m ... subsumes any of the others.

Considering again Example 3.7, Theorem 10 indicates that AL and ED are incommensurable with respect to their side effects.

b. Replacement

We first provide a theorem and an observation, which relate Type 1 side effects to the corresponding work on deletion.

Theorem 11. As with deletion, the root of a functionally determined subgraph will subsume the other relations in the subgraph, with respect to Type 1 side effects. (Note, however, that such action may not always be possible, depending on the state of the database.)

Proof: Analogous to the proof of Theorem 8. ∎

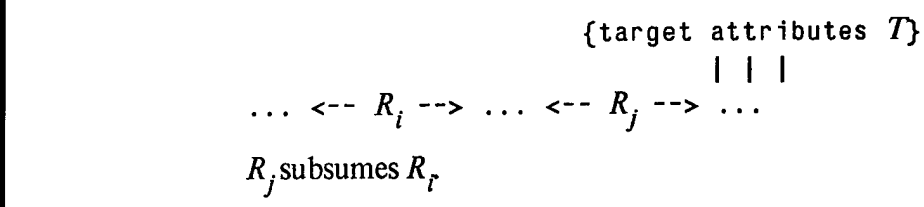
As with deletion, we can compare the side effects associated with different roots of the query graph. First, note that for Type 2 and Type 3 side effects, there will generally be at least one translation (the "direct" one), which has no side effects. This translation, however, may be blocked by constraints, necessitating use of an inexact translation. For Type 1 side effects, as with deletion side effects, there may be no exact translation. Theorems 9 and 10, which apply to side effects resulting from deletion requests, also apply to Type 1 side effects for replacement requests.

For Type 2 side effects. a different result is provided: such side effects are minimized by choosing the relation "nearest" the target, in the graphical sense.

Theorem 12. Let T be the set of target attributes for a replacement request and R_i, R_j be relations. If R_j separates R_i from the relation containing T, or R_j contains T, then R_j subsumes R_i, with respect to Type 2 side effects.

Proof: Type 2 side effects are associated with affected view attributes that are neither (a) mentioned in the update request, nor (b) accessible from attributes that do not change. The effects of condition (a) are the same for all translations. The effects of condition (b) are "monotonic"; as the total set of affected attributes gets larger, the set of affected attributes that are not determined cannot get smaller. All of the affected attributes that are associated with a change to R_j will also be affected attributes for a change to R_i (by definition of affected attributes, and the conditions of the theorem). Thus, the side effects for a change to R_i will be at least as large as for a change to R_j, and R_j subsumes R_i. ∎

The situation is:



R_j subsumes R_i.

Finally, we consider comparison of Type 3 side effects (insertion and deletion). As discussed, the preferred translations are those with no such side effects; Theorem 7 presented the conditions for this. Among translations that do have side effects, our first result indicates that, for a large class of requests, none of them will be superior to the others. Note that these situations and translations are more complex than are likely to be encountered normally; the discussion is included for completeness.

Theorem 13. If:

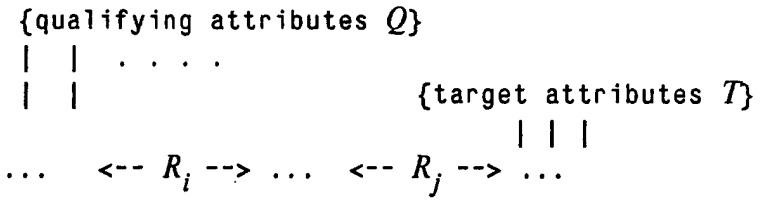
(a) all translations have Type 3 side effects (i.e., there is no translation such that $(R_C \cup E) \xrightarrow[V]{} A$), and

(b) some of the qualifying attributes Q are from a relation other than the one containing the target attributes T,

then all roots that separate the relation containing T from the relations containing any of

the attributes of Q (or that contain Q) will be incommensurable. in the sense that none will subsume any of the others.

The situation is as follows



Neither of R_i, R_j subsumes the other, with respect to Type 3 side effects.

Proof: To show that R_j does not subsume R_i:
From Lemma 2 and the conditions of the theorem, we know that a change to R_j has potential Type 1 side effects that a change to R_i does not. Thus, the translation will affect view tuples other than the marked ones.

From the nature of the translations, Type 3 side effects (insertions and deletions) will occur to these unmarked tuples, as well as to the marked tuples. Thus, the change to R_j will cause some potential insertions and deletions that are not associated with a change to R_i, and R_j does not subsume R_i.

To show that R_i does not subsume R_j:
Consider again the nature of the translations. If the original tuples are of the form: $(t_1, \ldots, t_i, \ldots t_j, \ldots, t_p)$, a change to R_j produces tuples of the form: $(t_1, \ldots, t_i, \ldots t_j^r, t_{j+1}' \ldots, t_p')$. Because $(R_j \cup E) \not\rightarrow A$, there will be Type 3 side effects, as explained in Theorem 7.

Now, a change to R_i produces tuples of the form: $(t_1, \ldots, t_{i-1}, t_i^r, t_{i+1}' \ldots t_j', t_{j+1}' \ldots, t_p')$. By Theorem 7, this may result in the creation of new generators of the form: $(t_1, \ldots, t_{i-1}, t_i^r, t_{i+1}'' \ldots t_j'', t_{j+1}'' \ldots, t_p'')$, t'' not necessarily all distinct from t', which correspond to extra tuples that would be inserted into the view.

Because $R_i \not\rightarrow R_j$ (by definition of root), by Lemma 1 t_j'' may be distinct from t_j'—i.e., extra generators may be created with varying values for R_j. But, if the replacement had been performed to R_j, the generators supporting the inserted tuples would all have the same value for R_j—t_j^r. Thus we have identified a set of insertions that would occur for R_i but not for R_j, and R_i does not subsume R_j. ∎

Remark: This is similar to Theorem 10, which states a similar condition for side effects for

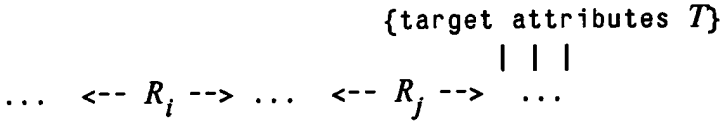
deletion requests. Essentially, each relation separates the other from some significant attributes: R_i separates R_j from Q, R_j separates R_i from T. Because of this, neither set of side effects is included in the other, and the relations are incommensurable.

This shows that, in general, Type 3 side effects are not easily compared. But, some of the side effects listed here are related to Type 1 side effects, caused by the arrangement of the qualifying attributes Q. If these effects are factored out, a comparison can be made, expressed in terms of the relative locations of the target attributes and root relations.

Terminology: Let Q be the qualifying attributes of an update request, Q' be that subset of Q (possibly empty) consisting of non-affected attributes. Now, for an insertion (or deletion) of a view tuple v' that constitutes a Type 3 side effect, if v' disagrees with the marked values for any attribute in Q', we refer to v' as a *modified* Type 3 side effect.

Theorem 14. Let T be the target attributes of an update request, R_i and R_j be root relations. If R_j separates R_i and the relation containing T, then when modified Type 3 side effects are ignored, R_j dominates R_i.

The situation described in the theorem is as follows:



Proof: We first show that every side effect that occurs with a change to R_j also occurs with a change to R_i, and then show that there are extra side effects associated with a change to R_i.

Consider the form of a change to R_i: a generator of a marked view tuple $(t_1, \ldots, t_i, \ldots t_j, \ldots, t_p)$ is replaced by $(t_1, \ldots, t_{i-1}, t_i^r, t_{i+1}', \ldots t_j', t_{j+1}' \ldots, t_p')$. If the change is made instead to R_j, the resulting generator is: $(t_1, \ldots, t_i \ldots t_j^r, t_{j+1}' \ldots, t_p')$.

Now, since the potential side effects associated with a given structural change are the same regardless of what value is used, consider the change to R_j which makes the t_n' tuples $(n=j+1, \ldots p)$ identical to the t_n' resulting from a change to R_i. Since the Type 3 side effects associated with a change to R_j are caused by tuples in the range $(R_{j+1}, \ldots R_p)$ (by Theorem 7), such side effects will also be associated with the change to R_i.

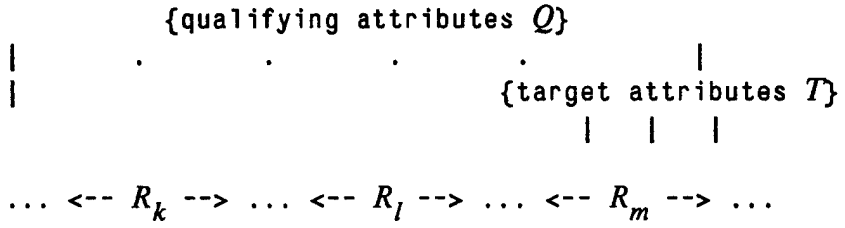
To see that the change to R_i causes extra Type 3 side effects, we use the fact the $R_i \not\rightarrow R_j$

and the same argument as in the second part of proof of Theorem 14. to show that the change to R_i will cause potential insertions that will not happen with the change to R_j. ∎

Remark: The above situation can be considered with respect to the graph. If the changed relation functionally determines the target attributes, there are no unmodified Type 3 side effects. Otherwise, each M:N relationship (shown graphically as dual "in" arrows at a node) between the changed relation and the target introduces another "level" of insertions and deletions.

Observations:

Consider a replacement request to be of the following form:



i.e., sets of qualifying attributes and target attributes specified in the update request, and a set of functionally determined subgraphs (each with a root), established by the cardinality of the relationships underlying the view definition.

Consideration of Type 3 side effects will dictate (by Theorem 7) that the translation be performed to a relation that functionally determines the affected attributes—i.e., a relation in the subgraph determined by R_m. Within this subgraph, none of the translations will have Type 3 side effects. Note that this group includes the direct translation, which is performed to the target attributes themselves. If these translations happen to be blocked by constraints, Theorem 14 will favor the subgraph "nearest" the target—i.e., R_l.

Consideration of Type 1 side effects will dictate (by Theorem 11) that the update be performed to the root of one of the subgraphs. In the typical case (qualifying attributes drawn from more than one underlying relation, as shown here), none of the roots will subsume the others (Theorem 10).

Consideration of Type 2 side effects will dictate (by Theorem 12) that the update be performed to the subgraph nearest the target attributes. Within that subgraph, the

translation nearest the targets—i.e., one acting on a leaf of the subgraph, away from the root—will subsume the others.

Re-examine Example 3.12.

```
ED  --->  DL ---> L  <--- AL; π: emp, adm
{ emp="Brown" ∧ adm="LaFleur" | adm←"Erman" }
```

There are two maximal functionally determined subgraphs:

```
ED ---> DL ---> L

L <--- AL
```

Consideration of Type 3 effects will favor the change to AL, which will have no such side effects. If this is blocked (e.g., if there is already an employee named "Erman"), one of the other translations (changing ED, or changing DL) will be chosen; these will have the same Type 3 side effects. Consideration of Type 1 will dictate that ED be favored over DL; ED and AL will not be comparable. None of the translations have Type 2 side effects. These results are consistent with the derivation of potential side effects performed originally. Thus, the update will be performed to AL, and failing that, to ED.

3.3.1.4. A note on multi-valued dependencies and side effects

Insight into the cause of some side effects can be gained by examining the associated query graphs. The problems with side effects are associated with nodes in the query graph having two "in" arrows. These points correspond to multi-valued dependencies (MVD's) in the view. Thus, in the model

```
ED ---> D <--- DM
```

The view has an (implicit) MVD constraint

```
dept ↠ emp | mgr
```

which is introduced and enforced by the view definition. Updates to the view cannot operate on single tuples, but must affect *sets* of tuples, to preserve the MVD. This occasions side effects, since an update request may attempt to change a single tuple. Thus, views that contain MVD's are not as amenable to updates as those with only functional dependencies. (A similar result was shown in [Keller, 1982].)

3.3.2. Pragmatic heuristics

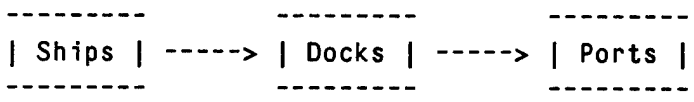
Certain kinds of low-level information that are present implicitly in the database schema may provide guidance for ranking updates. This section considers two types of information used by PIQUE.

3.3.2.1. "Static" vs "dynamic" attributes and relations

As part of the design process, some database systems distinguish between *dynamic* properties and relationships—which are expected to change frequently—and *static* ones, which are not. This distinction is typically used during implementation of the physical database, or during the design of update transactions. The information may also be used as the basis for a heuristic in ranking prospective translations:

- A change to a dynamic field or relation is preferred over a change to a static one.

Consider again Example 3.4 in Subsection 3.2.1: ships are associated with docks, and through them, with ports.



The ship-dock relationship is dynamic, whereas dock-port is static. Thus, a request to move a ship to another port would be fulfilled by reassigning the ship to a different dock, rather than by moving the current dock to a new port.

3.3.2.2. Changes to lexicon relations

Frequently, object clases may be identified by more than one property. For consistency, one of the properties will be used throughout the database, and a *lexicon* will be created to record the values of the other properties. For example, employees may be identified by name, employee-number, or social security number; these can all serve as *keys* of the employee relation. One of them (say, employee-number) will be selected to be used throughout the database, and the others will be recorded in the lexicon:

```
----------------------    ------------------------
| emp-no | dept | sal |   | emp-no | ss-no | name |
----------------------    ------------------------
```

The relation on the right is a lexicon.

This provides the basis for another heuristic:

- A change to a lexicon relation is less appropriate than a change to a non-lexicon relation.

This is motivated by the fact than an update to a lexicon is, in a sense, a name change for an entity; while such a change is possible, other methods of performing the update should be preferred.

Example 3.13:

Consider a revised form of the office domain, in which managers are identified by number, with the name recorded in a separate lexicon.

```
------------      ---------------      -------------------
|emp | dept| ---> |dept | mgr-no| ---> |mgr-no | mgr-name|
------------      ---------------      -------------------
```

and the usual request to change Brown's manager from Jones to Baker.

One translation of the request would be a change directly to the manager-name field of the appropriate tuple in the lexicon. While literally fulfilling the request, this translation seems inappropriate. PIQUE would rank this translation lower than the others, using this heuristic.

3.3.3. Semantic heuristics

These are heuristics motivated by the semantics of the domain modeled by the database. Information about the domain is typically represented in the *database schema.* The schema is expressed using a *data model,* which serves a function analogous to that of a *knowledge representation language* in an artificial intelligence system, providing a language for encoding information. Traditional data models (e.g. [Date, 1977]) have emphasized description of the structure of the *data*; more recent data models (e.g., [Nijssen, 1976]), reflecting the influence of contributions from artificial intelligence, provide facilities for describing the *domain.*

Semantic heuristics provide an accurate basis for ranking translations, since they reflect properties of the domain. However, most data models do not provide extensive semantic information, and the effort of encoding such information may be excessive. This section discusses four possible types of semantic information, of varying complexity; only the first two are expressed in the PIQUE data model (the Structural Model), and are used by the system. The others are beyond the expressive capabilities of the Structural Model, but are included for completeness.

3.3.3.1. Cardinality and dependency

On the basis of this information, certain updates may be rejected, because they would leave the database in an inconsistent state. The use of cardinality and dependency information is discussed, along with other integrity constraints, in the next section.

3.3.3.2. Abstraction vs. aggregation

Some data models ([Hammer and McLeod, 1978], [Wiederhold and El Masri, 1980], [Smith and Smith, 1977]) designate certain relationships as either *abstractions* or *aggregations*. An abstraction is a *generalization* of another entity (e.g., "ship-classes" is an abstraction of "ship"), whereas an aggregation is a homogeneous *collection* (e.g., "convoys" would be an aggregation of ships).

This distinction is rarely made, because there is no structural difference between the representations of the two types of relations (e.g., both ship-classes and convoys would be represented as separate relations, connected to the ships relation). However, there is a semantic difference with respect to updates: a change to an abstraction relationship is unlikely, whereas a change to an aggregation relationship is reasonable. PIQUE incorporates this distinction as the heuristic:

- Resist changes to abstraction relationships.

Thus, it is reasonable for a ship to be assigned to a new convoy, but less reasonable for the ship-class to change.

3.3.3.3. Likelihood estimates

Databases traditionally encode constraint information on a true-false basis, indicating whether a particular database state or state transition is valid. In certain situations it may be useful to assign numeric or symbolic *likelihood* estimates to such information, indicating the strength of the information concerned. This provides a measure of "common sense reasoning", as the term is used in artificial intelligence. We present two examples of use of such estimates in ranking updates.

(a) strength of connections and attributes

Connections in the database schema correspond to relationships between entities in the database. One can assign a strength to individual connections, indicating the resistance of the connection to change (i.e., the value could be inversely related to the frequency with which changes occur to the connection or attribute in question). Under such a scheme, a change to the cargo of a ship would be indicated to be reasonable, but a change to the name, or country of registry, would be less likely. Similarly, an attempt to move an employee to a new location would probably be executed by assigning him to a new department, rather than changing the location of the current department. This scheme is a generalization of the distinction between static and dynamic attributes, discussed in Subsection 3.3.2.

(b) permanence of entities

Certain classes of entities may be more transient than others, and thus more susceptible to insertion, deletion, or modification. Thus, in a library database, the catalogue record for a particular book would be relatively permanent, whereas the loan record indicating that the book is borrowed by a particlar customer would be updated frequently.

The heuristic associated with this information is:

- Prefer those updates that have higher likelihood.

For both types of likelihood information, the numeric values would be assigned in advance by the database administrator. These values could typically be formed analytically, as part of the process of database design, but might also be derived empirically, from observation of users' interactions.

Numeric likelihood estimates have been used in artificial intelligence, in applications such as the *certainty factors* of the MYCIN system [Shortliffe, 1976] or fuzzy logic [Zadeh, 1982].

3.3.3.4. Process models

A more effective form of semantic support for updates is achieved by modeling the kinds of changes that can occur in the domain that the database models. Current data models establish a correspondence between states of the domain and states of the database; for updates, this may be extended to establish a correspondence between transitions in the domain and transitions in the database. This notion is derived from work in databases (transition constraints [Date, 1983]; transactions and super-transactions [Date, 1983]), and artificial intelligence (scripts [Schank and Abelson, 1977]). A description of a proposed system using this method to interpret natural language updates is provided by Maier and Salveter [Maier and Salveter, 1982].

The *process models* would associate, with each class of change in the domain, information such as:

- –attributes affected by the change;
- –preconditions for the change;
- –an estimate of likelihood of such a change;
- –associated changes that may be required;
- –specification of default values;
- –questions that may be asked of the user to clarify ambiguities.

Process models may be low-level, corresponding to single database changes (e.g., changing the cargo of a ship), or high-level, corresponding to complex actions in the domain (e.g., hiring a new employee).

Use of such models changes the essential character of update interpretation. It provides increased accuracy and the ability to handle complex requests, at the expense of requiring a great deal of semantic information, including anticipation of all possible updates. This approach is discussed further in Section 6.1.1.

3.3.4. Integrity constraints

In addition to a description of the structure of the database, the database schema may contain *semantic constraints*, which circumscribe the set of valid states (extensions) of the database. These include restrictions such as:

- –functional dependencies (e.g., "Each employee has a unique social security number.");
- –reference constraints (e.g., "No employee can be assigned to a department that doesn't appear in the Departments relation.");
- –domain definitions (e.g., "An employee's age must be a number between 0 and 100.");
- –multi-valued dependencies (e.g., "Textbooks are assigned to courses, independent of the teacher handling the course.").

During the process of interpreting an update request, these constraints may serve to eliminate certain translations—those which would leave the database in an invalid state. PIQUE implements the first two types.

These constraints are used during the candidate ranking process, together with the following heuristic:

- Updates that violate semantic constraints (more precisely, those that produce database states that violate semantic constraints):
  - (a) are ranked lower than translations that do not violate such constraints;
  - (b) are never performed, even if they have the highest ranking.

Semantically invalid translations are not simply discarded, because they may be useful for generating an explanation for the user about the failure of his request. (See Section 3.4, "Action Taken").

Example 3.14:

Consider the EM view of Example 2.1, and a request to remove the tuple (Brown, Jones). One translation is to delete the tuple (Sales, Jones) from the DM relation. This would violate the reference constraint (the employees in the Sales department would no longer have a corresponding department record) and would therefore be rejected.

Example 3.15:

For an example demonstrating the use of a different form of constraint, not implemented in PIQUE, consider a revised version of the EDM database:

emp	ctzn	dept
LeBrun	Fr.	Sales

dept	mgr
Sales	Jones
Security	Baker

and the usual request, to change LeBrun's manager to Baker.

As before two translations exist. If, however, there is a semantic constraint that employees of the security department must be American citizens, translation (b), which entails of moving LeBrun to a new department, would violate this, and must therefore be rejected. (This form of multi-attribute restriction is referred to as a *production* in [King, 1981].)

3.3.4.1. Constraint checking

If the constraints are of certain standard types (specifically, functional dependencies and reference constraints), checking for constraint satisfaction may sometimes be done by examining only the schema and update, without reference to the database extension. This is analogous to the use of *potential*, rather than *actual* side effects, as described in Subsection 3.3.1.

In this section we present conditions used by PIQUE to determine potential constraint violations for translations. We assume that the only integrity constraints are those described in Subsection 3.3.1—functional dependencies and reference constraints that are encoded as part of the structural model description of the database. (For other types of constraints, checking must usually be performed extensionally.) Deletion and replacement are examined separately.

a. Deletion

(1) A deletion cannot cause violation of a functional dependency.

(2) A deletion may violate a reference constraint. Specifically, there will be a potential constraint violation if the relation to which the deletion is performed is *referenced* by another relation. This condition is indicated in the query graph by an arc pointing in to the changed relation; the relation is not a *root* of the graph.

b. Replacement

There are two replacement cases, depending upon whether or not the update includes a change to a *join* attribute.

(1) If the change(s) are to non-join attributes:

—A change to a dependent (non-key) attribute cannot violate a functional dependency. A change to a key attribute may cause such a violation; this will occur if the new value concides with the key of an existing tuple. This condition must be checked extensionally.

—A change to a non-join attribute cannot violate reference constraints.

(2) If the change is to a join attribute:

For views restricted to the relationships of the structural model, only certain updates will be valid. In particular, only the head of an arrow may be redirected, by changing the referencing relation. Changes of this form cannot violate functional dependencies or reference constraints.

Example 3.16:

```
ED --> DM
{  emp="Brown" : mgr←"Baker" }
```

The translation which changes the link (reassigning Brown) is semantically acceptable.

Changes that violate this restriciton (i.e., by changing the referenced end of a link) will violate reference constraints, by leaving some tuples "dangling". They may also violate a functional dependency in the referenced relation.

Example 3.17:

```
ED --> DM
{ mgr="Jones" : emp←"Adams" }
```

A change to the link (by changing the department of Jones), will leave some ED tuples dangling, in violation of the reference constraint. If the department to which Jones is assigned already has a manager, there will also be violation of a functional dependency,

3.3.4.2. Integrity constraints and instantiation

Indirect translations, in which a link is redirected to effect a change to the value of a different attribute, are *contingent* upon the presence of certain tuples in the database, as described in Section 3.2.6.2. Certain conditions may exist that will eliminate these translations without the necessity of searching the database:

(1) Let T be the target attribute of the update request, Q the qualifying attributes, and Q' be the subset of Q (besides T) that are also affected attributes for the candidate translation in question. Similarly, define E, E', for equi-selected attributes. If $(Q' \cup E') \xrightarrow[V]{} T$, there is no consistent initial state from which the update may be performed in this way.

This follows from the fact that none of the attributes in $(Q' \cup E')$ are changed by the update (by definition of candidate translation) which implies that T cannot change.

Example 3.18:

```
ED --> DM --> ML
{ emp="Brown" ∧ mgr="Jones" : loc←"NY" }
```

Consider a translation that changes the link between ED and DM. For this translation, $Q'=\{\text{mgr}\}$, $T=\{\text{loc}\}$, $Q' \xrightarrow[V]{} T$. No change to the link will perform the update, since there cannot be another tuple of the ML relation that has Jones in NY.

(2) Let C refer to the affected attribute that is involved in the link being changed. If $(Q' \cup E') \xrightarrow[V]{} C$, there is no consistent initial state from which this translation will perform the update. This follows from the fact that the attributes in $(Q' \cup E')$ do not change (as shown above), and hence C cannot change.

Example 3.19:

```
ED --> D <-- DM <-- ML
{ emp="Brown" ∧ mgr="Jones" : loc←"NY" }
```

Consider a change to the link between ED and D. $Q'=\{\text{mgr}\}$, $C=\{\text{dept}\}$, $Q' \xrightarrow[V]{} C$. Again, no change will work, because all tuples with manager Jones will have the same department value.

Note the difference between the application of integrity constraints here and in Examples 3.14 and 3.15. In previous cases, constraints were used to rule on the validity of database resulting from the update. Here, they are used to prove that the appropriate states cannot exist before the update.

3.3.5. Order of application

The PIQUE ranking heuristics are applied in the following order:

(1) Eliminate translations that violate integrity constraints;

(2) Prefer translations favored by static/dynamic considerations;

(3) Prefer the translations favored by accuracy considerations (i.e., those with minimal potential side effects);

(4) Prefer translations that are favored by semantic considerations;

(5) Prefer translations favored by other pragmatic considerations;

(6) Prefer translations involving the minimal number of changes to the database.

The heuristics at each level are applied only if ambiguities remain after application of the heuristics from the level above.

It must be stressed again that this ordering is particular to PIQUE; in different circumstances, a different ordering or a different set of heuristics may be appropriate. The current set of heuristics has been selected on the basis of availability and portability. Such choices are discussed further in Chapter 5.

3.4. Action Taken

Once the candidate translations have been generated and ranked, PIQUE takes action based on the resulting list. Ideally, one of the translations will outrank the others, and the system will execute that translation. This may not be possible in general; some more complicated resolution algorithm may be required. The algorithm used by PIQUE can be summarized by:

(1) If one candidate outranks the others by a sufficient margin, execute it, notifying the user of any side effects that occur;

(2) If two or more candidates are roughly equal in ranking, present them to the user and ask which was intended;

(3) If there is not an acceptable candidate, but there are one or more whose execution is blocked by semantic constraints, present them to the user, with an explanation as to why they cannot be used.

We examine these operations in detail. Note that the three phases share a common need to present information to the user. Examples of responses are presented in the individual discussions, and the problem of response generation is considered in a separate section.

3.4.1. Executing one of the candidates

If one candidate outranks the others, it is executed. To minimize error, a margin of difference is required between the leading candidate and the next best. This margin is embodied in the ordering of heuristics. Differentiations made by the top three classes of heuristics (constraints; static/dynamic distinctions; accuracy) are considered strong enough to warrant performance of the top candidate, whereas the others are too weak, and provide only an ordering for presentation purposes.

Ideally, the chosen candidate will exactly fulfill the user's request; however, it may have side effects. In this case, the user is notified of the side effects, to prevent future surprises. The presentation of side effects is important, because it alerts the user to possible ramifications of an update. He may understand the operation, without being aware of its associated effects on his view. Thus in Example 2.1, even if the user knows that the manager of the sales department is replaced, he may not know that Smith and Palen work in that department and will be affected by the update.

3.4.1.1. Notification of side effects

Each type of side effect is presented to the user in a different way, using a predefined template. The examples below are indicative of side-effect responses; in each case, the user has asked for a change to the view tuple for a certain employee.

(a) deletion: "Note that the tuples (Smith, Jones) and (Palen, Jones) have also been deleted."

(b) replacement, type 1: "Note that the tuples (Smith, Jones) and (Palen, Jones) have also had the value of the attribute Mgr changed."

(c) replacement, type 2: "Note that the value of the attribute Location has also been changed."

(d) replacement, type 3: "Note that extra (Emp. Adm) tuples have been inserted: (Brown, Quin) (Brown, Erman)".

3.4.2. Asking the user to choose among a number of potential candidates

If no translation is clearly superior to the others, the user is presented with the best candidates and asked to choose among them. The candidates are described, when possible, in terms familiar to the user. The use of such a *clarification dialogue* to resolve ambiguities is similar to the approach of the *RENDEZVOUS* system [Codd et al, 1978], although the version used in PIQUE is much simpler.

Example 3.20:
In response to a request to change Brown's manager, the user might be presented with a choice like:

```
This update could be performed in the following ways:
    by changing the DEPT attr of the tuple
        EMP     DEPT
        BROWN   SALES
to either INVNTRY or SECURTY.  Which would you prefer?
```

Although the difference in ranking between the candidates is insufficient to warrant performance of one of them, there may be slight differences. In such a case, the candidates are presented to the user in descending order of ranking.

As part of the presentation of translations in a clarification dialogue, it may be desirable to list the side effects associated with each translation, to indicate the implications of the change. Instead of presenting the actual side effects, which requires reference to the database extension, an alternative is to present the potential side effects, which are derived using the algorithms of Subsection 3.3.1. Two examples of this form are:

—deletion: "may also delete other tuples, with the wrong values for the attribute Emp." (That is, may delete other employees besides the requested one(s).)

—replacement, type 2: "may also change the value of Loc for this Emp."

3.4.3. Explaining a failure due to violation of semantic constraints

If the best candidates are blocked by constraints associated with the database, the request is *impossible* and cannot be fulfilled. The appropriate response in this situation is an explanation of the reason for the failure. Note that these are in a sense contingent failures, caused by a particular state of the database. The user must be notified of:

(a) what constraint was violated;

(b) what aspect of the database extension caused the violation.

Note that the user may be unaware of either, or both, aspects of the problem: he may not understand the ramifications of his particular update request, or he may not have known about the constraint in question.

In some cases it would be possible to restore consistency to the database by changing an attribute of a non-involved tuple. We reject these update translations. Our approach has been to notify the user of the problem, to allow him to rephrase the request himself.

For certain classes of semantic constraints, including the ones implemented in PIQUE, the reason for failure can be easily explained. We present examples of the appropriate responses for each type of constraint violation.

3.4.3.1. Functional dependencies

Example 3.21:

```
"This update could be performed by setting the EMPNO attribute
     of the tuple
          EMP      EMPNO
          SMITH    222
 to 103.  However, that value has already been assigned
     to the tuple
          EMP      EMPNO
          ADAMS    103

 This update would violate the functional dependency
          EMPNO -> EMP.
 (That is, each EMPNO must be assigned to a unique EMP.)"
```

3.4.3.2. Reference constraints

Example 3.22:

```
"This update could be performed by setting the DEPT attribute
     of the tuple
          EMP      DEPT
          BROWN    SALES
 to MKTING. However, this cannot be done, because there is no
 DM tuple with NAME equal to MKTING.  This violates the constraint
 that all EMPLOYEES must have a DEPT attribute which appears as
 a NAME in the DEPT relation."
```

3.4.3.3. Other constraints

There are two other types of constraints, not currently implemented in PIQUE, which could be handled with only minor extensions:

(a) domain definition [McLeod, 1976];

This includes range constraints, such as

```
deadweight ε [20,450]
```

The response to a violation of this type of constraint would be something like

```
"This update could be performed by setting the DW attr
     of the tuple
          SHIP     DW
          KRANJ    250
 to 600, which would violate the constraint that for all SHIPS
 the DW must always have a value between 20 and 450."
```

(b) bounding rule [King, 1981];

This includes constraints in which the value of one attribute is determined by the value of another; for example,

```
cargo.quantity < capacity
```

The explanation required for such a constraint would be similar to the previous case. Current database systems do not generally support this type of constraint, though.

3.4.4. Response generation

Natural language database systems must sometimes provide responses that involve information from the database extension. PIQUE, for example, provides information about (a) actual side effects, (b) candidate translations, and (c) constraint violations, all of which require presentation of a particular set of database tuples. In many instances a simple list of the relevant tuples is inappropriate. This subsection briefly examines the range of potential responses, and indicates PIQUE's capabilities.

The response generation problem bears some resemblance to the view model problem discussed in Section 3.1. The goal in both situations is to represent a segment of the database, in a way that presents the appropriate aspect.

For such information, three *levels* of presentation can be identified:

(a) extension;

(b) intension;

(c) description.

The *extension* of a set of tuples is the set itself. The tuples to be changed by a candidate translation might be listed as "Brown, Adams, LeBrun, . . .". The *intension* is the defining expression for that set. For the set in question, this might be "The employees in the sales department." A *description* for a set of tuples is based on properties not necessarily restricted to those in the intension. A description might be "Five employees who live in San Diego." The description thus involves *assertional* rather than *structural* properties of the set [Woods, 1975]. The appropriateness of each level is dictated by the situation. Most database systems provide only extensional responses, and the examples in this thesis are mainly extensional. However, such a response is inappropriate if the set is large; a list of 3,000 employees may not be very helpful to the user.

Intensional responses are potentially useful, but are inappropriate in a variety of cases:

(a) The expression defining the target set may be unavailable; the response must be derived from the extension alone.

(b) The intension may not be informative. To a question, "Which employees profit-share?" a reply of "those that profit-share" is not useful.

(c) The intension may be more complex than the extension. A set of employees may

be described as "those employees who work in technical departments located in New York" or "Brown and Foster".

Descriptions are formed by induction over the target set. The group of properties to be used for the induction is determined by the situation and the database extension; *salient* properties (e.g., keys, qualifying attributes, join attributes) are preferred. Salience may also be determined by the user model. In addition, a count of the tuples may be useful.

If the tuples have many attributes, another form of descriptive response is required: a subset of the attributes must be selected for presentation to the user. Typically, only salient attributes are presented.

PIQUE provides extensional responses, and descriptive responses of the second type (presenting selected attributes). Implementation of the remaining types of responses is not conceptually difficult, but is beyond the scope of this thesis.

Chapter 4
Examples

In this section, we present a number of examples of interactions with PIQUE.

The PIQUE system is implemented in Interlisp on the DEC-20 at SRI International. The natural language interface uses the LIFER package, developed by Gary Hendrix [Hendrix, 1977]. The database interface uses SODA, a LISP-compatible relational calculus-like language designed by Robert Moore [Moore, 1979]. The SODA interpreter, written by Bil Lewis, has been extended to handle updates. The database system used is a LISP-embedded relational database.

The examples will be taken from a business/personnel domain, consisting of the following relations. Key fields of relations are shown on the left side, separated from non-key fields by the "||".

```
Employees
        ------------------------------
        | emp || empno | sal | edept |
        ------------------------------

Departments
       ----------------------
       | dept || mgr  | dloc |
       ----------------------

Locations
       -------------
       | loc  || vp |
       -------------

Administrators
       ----------------
       | admin | aloc ||
       ----------------
```

Employees are grouped into departments, which are assigned locations. Each location has a vice president in charge of it, and a group of administrators which serve all the departments based at that location. In the structural model, the relation connection graph for this database appears as:

```
Employees ---> Departments --> Locations <--- Administrators
```

The Employees and Departments relations are connected via a join on the Edept and Dept attributes; Departments are connected to Locations, via Dloc and Loc; Administrators are connected to Locations, via Aloc and Loc.

We assume initially that there are no constraints besides those implicit in the structural model connections, and the dependencies associated with keys.

The initial database extension is:

Employees

Emp	Empno	Sal	Edept
Adams	103	30	Invntry
White	431	35	Mkting
Brown	554	30	Sales
Smith	222	25	Sales
Palen	181	25	Sales

Departments

Dept	Mgr	Dloc
Sales	Jones	SF
Mkting	Baker	LA
Invntry	Fisher	SD
Secrty	Moseley	SD
Advert	Larkin	NY

Locations

Loc	VP
SF	Nixon
LA	Loftus
SD	Wing
NY	Walsh

Administrators

Admin	Aloc
Lafleur	SF
Shutt	SD
Olson	SF
Erman	NY
Quin	NY

Some of the examples contain "trace" information, displayed to show the internal states of the program; this information is indented. Additional annotation is enclosed in brackets.

4.1. Example 2.1, Revisited

```
Q1:>(LIST THE EMPLOYEES AND THEIR MANAGERS)

EMP      MGR
-------------
ADAMS    FISHER
WHITE    BAKER
BROWN    JONES
SMITH    JONES
PALEN    JONES

Enter next command:

Q2:>(CHANGE BROWN'S MANAGER FROM JONES TO BAKER)
```

[The system provides some "trace" information, listing the candidate translations. There are two of these.]

```
        The possible ways of performing the update:

        1.  In the relation DEPARTMENTS change the MGR attribute
        of the tuple

        DEPT     MGR
        ------------------
        SALES    JONES

        to the value BAKER

        2.  In the relation EMPLOYEES change the EDEPT attribute
        of the tuple

        EMP      EDEPT
        ----------------
        BROWN    SALES

        to the value MKTING
```

[The translations are now ranked, using the heuristics. For this case, the accuracy heuristics are sufficient to distinguish between the candidates.]

```
        These translations have the
        following effects:

        1. Effects are:
        In the view: potentially changing the MGR of other EMPLOYEES.

        2. Effects are:
        None.

        Desired translation is: 2.
```

```
Revised view is:

EMP     MGR
-------------
ADAMS   FISHER
WHITE   BAKER
BROWN   BAKER
SMITH   JONES
PALEN   JONES
```

[We observe that the user's request has been fulfilled exactly.]

4.2. Example of Notification of Side Effects (Type 1)

```
Q1:>(LIST THE EMPLOYEES AND THEIR MANAGERS)

R1:      EMP      MGR
         -------------------------
         ADAMS    FISHER
         WHITE    BAKER
         BROWN    JONES
         SMITH    JONES
         PALEN    JONES

Q2:>(CHANGE BROWN'S MANAGER FROM JONES TO CHOATE)
```

[Note that Choate is not currently the manager of a department, so the option of moving Brown to a new department is not available. The only translation is the direct one, of changing the manager of Brown's department. In this and subsequent examples, translations, when they are presented, will be shown in the internal language used by the system.][3]

```
Translations:

1: ((IN Y DEPARTMENTS) (SOME (IN X EMPLOYEES)
                             ((X EMP) EQ 'BROWN')
                             ((Y DEPT) EQ (X EDEPT)))
    ((Y MGR) ← 'BAKER'))
```

---

[3] A note on the syntax of SODA: the SODA forms used here can be mapped easily into a calculus-based DML like the one used in this thesis:

```
(IN X EMPS)                          => X ∈ EMPS
((X EMP) EQ 'BROWN')                 => X.EMP = BROWN
(SOME (IN X EMPS)(IN Y DEPTS) ...)   => ∃ X ∈ EMPS, Y ∈ DEPTS ...
```

```
R2: Done

Revised view is:

EMP      MGR
-------------
ADAMS    FISHER
WHITE    BAKER
BROWN    CHOATE
SMITH    CHOATE
PALEN    CHOATE

Note that the tuples for employees SMITH and PALEN have also had the
value of the attribute MGR changed to CHOATE.
```

[The system notifies the user of the side effects that have resulted from the update.]

4.3. Example of a Genuinely Ambiguous Request

```
Q1:>(WHERE ARE ALL THE EMPLOYEES LOCATED)

R1:      EMP      LOC
         --------------------
         ADAMS    SD
         WHITE    LA
         BROWN    SF
         SMITH    SF
         PALEN    SF
```

[Note that the employee names are included in the response, although they have not actually been requested. This is an example of a *supportive indirect response* [Kaplan, 1979]. PIQUE provides such responses in certain cases.]

```
Q2:>(BROWN HAS NOW MOVED TO SAN DIEGO)

R2: This update can be performed in the following ways:

    By changing the EDEPT attribute of the tuple

        EMP      EDEPT
        --------------------
        BROWN    SALES

    to either INVNTRY or SECURTY.

Which would you prefer?
```

[The system has eliminated one translation—that of moving Brown's department to San Diego—because the location of departments is "static". Two translations remain. With no basis for choosing between them, the system must consult the user.]

4.4. Example of an Impossible Request

[Consider an additional constraint, the functional dependency EMPNO -> EMP (i.e., make Empno a key of the Employees relation).]

```
Q1:>(WHAT ARE THE EMPLOYEE NUMBERS FOR EMPLOYEES IN THE SALES
     DEPARTMENT)

R1:      EMP      EDEPT    EMPNO
         ---------------------
         BROWN    SALES    554
         SMITH    SALES    222
         PALEN    SALES    181

Q2:>(CHANGE SMITH'S EMPLOYEE NUMBER TO 103)

         Translations:

         1. ((IN X EMPLOYEES)
             (AND ((X EDEPT) EQ 'SALES') ((X EMP) EQ 'SMITH'))
             ((X EMPNO ) ← 103))

         Ranking:

         1. Violation of functional dependency EMPNO -> EMP.
```

[This update cannot be performed, because it would violate the functional dependency. One possible response from the system is "No"; this is inappropriate at best. A more informative response is actually provided.]

```
R2: This update could not be performed, because of semantic
    constraints:

The EMPNO value of 103 has already been assigned to the tuple

         EMP      EMPNO
         ----------------------
         ADAMS    103

which has the EDEPT value of INVNTRY.

This update would violate the functional dependency
         EMPNO -> EMP.
(That is, each EMPNO must be assigned to a unique EMP.)
```

[Note that the EDEPT of the Adams tuple is printed. Without this, the user may not realize why he cannot see the tuple. This attribute is printed because of its *salience*—it appears in a selection in the view. The phrasing of the response is thus determined by the view model.]

4.5. Example of Side Effects (Type 2)

[Consider a different type of side effect, a change to an attribute that has not been mentioned in the update.]

```
Q1:>(LIST THE EMPLOYEES, THEIR DEPARTMENTS, AND THEIR MANAGERS)

R1:     EMP       EDEPT     MGR
        ----------------------------
        PALEN     SALES     JONES
        SMITH     SALES     JONES
        BROWN     SALES     JONES
        WHITE     MKTING    BAKER
        ADAMS     INVNTRY   FISHER

Q2:>(MOVE WHITE TO THE INVNTRY DEPARTMENT)

        Translations:

        1. ((IN X EMPLOYEES) (SOME (IN Y DEPARTMENTS)
                                   ((X EMP) EQ 'WHITE')
                                   ((Y DEPT) EQ (X EDEPT)))
            ((X EDEPT) ← 'INVNTRY'))
```

[The translation is the obvious one, changing White's department. The ranking process detects the possibility of side effects associated with this change.]

```
        Ranking:

        1. Type 2 side effects, change to MGR.

Revised view is:

EMP       EDEPT     MGR
----------------------------
PALEN     SALES     JONES
SMITH     SALES     JONES
BROWN     SALES     JONES
WHITE     INVNTRY   FISHER
ADAMS     INVNTRY   FISHER


Note that some extra changes have been made. Besides having the
value of the attribute EDEPT changed, the tuples in question have
also had the MGR attribute changed.
```

[The system performs the update, then reports the extra change to the user.]

4.6. Example Involving Type 1 and Type 3 Side Effects

[To see the complexity that can arise in the mapping from the natural language to the database level, independent of the language itself, consider an example that is syntactically identical to Example 2.1:]

```
Q1:>(LIST THE EMPLOYEES AND THEIR ADMINISTRATORS)

R1:     EMP       ADMIN
        --------------------
        PALEN     OLSON
        PALEN     LAFLEUR
        SMITH     OLSON
        SMITH     LAFLEUR
        BROWN     OLSON
        BROWN     LAFLEUR
        ADAMS     SHUTT

Q2:>(CHANGE BROWN'S ADMINISTRATOR FROM LAFLEUR TO ERMAN)

        Translations:

        1. ((IN W ADMINISTRATORS
                  . . .
            ((W ADMIN) ← 'ERMAN'))

        2. ((IN X EMPLOYEES)
                  . . .
            ((X EDEPT) ← 'ADVERT'))

        3. ((IN Y DEPARTMENTS)
                  . .
            ((Y DLOC) ← 'NY'))
```

[There are 3 translations. The first is the direct one. The second involves reassigning Brown. The third involves changes to the location of departments.]

```
        Ranking:

        1. Type 1 side effects.
```

[may change the manager of other employees]

```
        2. Type 3 side effects.  Type 1 side effects.
```

[may insert/delete tuples for other administrators of this employee]

```
        3. Type 3 side effects.  Type 1 side effects.  Change to
           static field.
```

[superset of the side effects for (2)]

[The actual side effects are:

1. will also change the manager of Smith and Palen;
2. will also delete (Brown, Olson), and insert (Brown, Quin);
3. will have the same side effects as (2), and also delete (Palen, LaFleur) (Palen, Olson) (Smith, LaFleur) (Smith, Olson), and insert (Palen, Erman) (Palen, Quin) (Smith, Erman) (Smith,Quin).]

```
Revised view is:

EMP       ADMIN
--------------------
PALEN     OLSON
PALEN     ERMAN
SMITH     OLSON
SMITH     ERMAN
BROWN     OLSON
BROWN     ERMAN
ADAMS     SHUTT

Note that SMITH and PALEN have had the value of the ADMIN
attribute changed to ERMAN.
```

[Of the translations, the system preferred the one that had only type 1 side effects, and thus avoided insertions and deletions. This is arguably the most intuitive response to the request.]

Chapter 5
Evaluation

In this chapter we evaluate the efficacy of PIQUE as a natural language update system. The first section is a discussion of *coverage*: what classes of updates are handled. The next section is a discussion of *correctness*, including a classification of potential errors and methods for their prevention. Next are presentations of *efficiency* and *portability*.

5.1. Coverage

Coverage concerns the range of the capabilities provided by the system and its natural language interface. We distinguish *linguistic* coverage—the range of linguistic phenomena handled by the system—from *logical* coverage—the range of domain capabilities that can be performed using the natural language front end.

Linguistic capabilities have not been stressed in PIQUE, and linguistic coverage is therefore quite limited. Many linguistic phenomena—ellipsis, relative clauses, conjunction, passive voice—are handled only in simple cases, if at all. Extension of the system to fully handle these could be accomplished via expansion of the grammar; this was not the focus of the PIQUE research.

Logical coverage (related to *conceptual* coverage, as defined by Tennant [1980]) concerns the classes of requests and activities that can be expressed via the natural language interface. For many applications, this cannot be effectively characterized, because the range of potential actions is large or unstructured. For database interfaces, however, the range is well-defined, specifically by the capabilities of the underlying data manipulation language. PIQUE's logical coverage can be precisely characterized:

(a) The system handles database queries and updates (as opposed to, for instance, meta-queries, about the structure of the database).

(b) For both queries and updates, the set of requests handled is those whose expression in the DML has certain properties. Specifically, the qualification must be a conjunction, and the set of joins involved must define chain. Sections 3.1 and 3.2 explain this in greater detail.

Coverage leads to an associated concept, *completeness*, which concerns the degree to which the system covers the total domain of possible requests. Again, the completeness of database systems can be formalized. For PIQUE, total completeness consists of the ability to handle all queries and updates expressible in the relational calculus. A more useful form, deriving from Tennant's version of completeness, relates the coverage of the system to the total coverage desired by users—i.e., how well the system meets users' (observed) needs. PIQUE's update capability cannot be evaluated in this latter way, since no natural language update systems have previously been tested. However, the reasonableness of the restriction to conjunctive chain qualifications for queries and updates is supported by such work as [Thompson, 1981], who presents a sample of 882 actual natural language queries, *all* of which meet this restriction. These and other data suggest that the full capabilities of data manipulation languages are infrequently used and that conjunctive requests are a natural and adequate sub-class. of queries. Many of the restrictions on PIQUE's coverage are motivated by convenience; extension of the coverage is complicated, but not conceptually difficult.

5.1.1. Insertion

Of the three types of update operations possible for a database system, PIQUE handles two. The performance of *insertions* is excluded, for a number of reasons:

(a) Unlike deletion and replacement, insertion typically requires updating multiple relations in the underlying database;

(b) The number of candidates is often large, and sometimes infinite; the candidate generation algorithm is more complex than those for deletion or replacement;

(c) To perform insertion in other than simple cases, mechanisms beyond the current PIQUE framework are necessary.

This subsection considers the problems of insertion in greater detail and discusses a variety of approaches having different power.

The main problem with insertion through any view mechanism concerns the *join* attributes of the view. If values for these attributes do not appear in the view (and thus are not specified in the update request), *marked nulls* (also called *Skolem constants* in formal logic work) must be introduced ([Reiter, 1983], [Maier, 1980]). For example, to insert a tuple (Clark, Butkus) into the *EM* view, a Skolem constant ω_1 is created, and two tuples are inserted into the database:

Emp	Dept
Clark	ω_1

Dept	Mgr
ω_1	Butkus

The marked null is really a shorthand for an existentially quantified variable. Use of constructs of this form thus renders the database incomplete, and introduces a number of problems, including the need for a truth value "unknown". The subject of such incomplete databases is discussed further in Subsection 6.1.2.

An alternative to the use of marked nulls is to allow only views in which all join attributes are visible. Under such a constraint, insertion becomes tractable. The update request specifies exactly which tuples are to be inserted in the database; there is only one candidate. For each database tuple to be inserted, either:

(a) the tuple may safely be inserted; the system will do so;

(b) the tuple is already present in the database extension; the system will continue;

(c) the tuple would violate a functional dependency; the system will fail.

This approach, used in [Keller, 1982], avoids the problem of incompleteness, but severely narrows the class of acceptable views. Note that, although marked nulls are avoided, nulls are still allowed in non-join fields; these nulls cause other problems, as discussed in [Date, 1983].

A related approach is to allow views with invisible join attributes, but to question the user upon each insertion request, to establish the value of the join attributes. The method is otherwise the same as the one above. The effect is to force the user to specify the complete translation of his request.

In certain circumstances, the value of join attributes may be derivable from the current contents of the database. Given a database extension:

Emp	Dept
Adams	Invntry
Brown	Sales

Dept	Mgr
Invntry	Fisher
Sales	Jones

and a request to insert a tuple (Brenly, Fisher), a reasonable translation is to insert a database tuple (Brenly, Invntry). In general, such an update is possible only if the missing tuples are members of *root* relations of the query graph corresponding to the view (in Structural Model terminology, if the only insertions are to *primary entity* relations). This is another tractable class of natural language insertions.

Finally, the problem may be addressed with domain semantic knowledge. Insertion, more than deletion or replacement, benefits from semantic information. In particular, super-transactions, default values, clarification dialogues, etc., all reduce the complexity of the problem. Natural language may prove too verbose in general to be used for insertion, unless complex semantics can be specified for domain terms such as "hire" or "schedule". This *knowledge-based* approach to updates is discussed in Subsection 6.1.1; the requirement of specification of large amounts of domain information is contrary to the PIQUE approach.

In summary, performance of natural language insertions requires either more powerful mechanisms (e.g., incomplete databases, semantic world knowledge) or restrictions on the problem space.

5.2. Correctness

A system that attempts to behave "intelligently" on ill-structured problems introduces the possibility of error. This section discusses how *well* PIQUE performs updates. We consider the errors that can be introduced in each phase of the system and how they might be avoided. (The error avoidance methods presented below have not yet been implemented in PIQUE.) The analysis of PIQUE's behavior is complicated by the fact that the "right" answer to an update request is not always evident. The appropriate response may depend on the user, the nature of the system, or other factors.

5.2.1. Problems in modeling the user view

If the model of the user's view is wrong, the system will misinterpret the basic intent of a user's request. This can generally be avoided by establishing careful conditions for application of the view model (as discussed earlier) or by using feedback to make the user aware of the view assumption.

As discussed in Section 3.1, the view model contains two main types of information about the user's current focus: (a) restrictions on sets of entities, and (b) paths to be used for navigation, corresponding to selections and joins, respectively, in the view expression. For the first type, restriction, feedback can be used if necessary to indicate the system's interpretation:

"By 'Jones', I assume that you mean the Jones who is a programmer."

A view expression that contains an incorrect selection operation will typically cause the update to be applied to too broad or too narrow a set. This type of error is thus similar to the Type 1 side effects (and Deletion side effects) discussed in Subsection 3.3.1.

The second type of view information—navigation—is a common area of difficulty in database systems with high-level interfaces (e.g., LADDER [Hendrix et al, 1978]; System U [Korth and Ullman, 1980]). In many cases, the path chosen will be the only possible one. For others, feedback will be appropriate:

"I assume that you mean the professor who is the supervisor of White."

The classification and avoidance of errors in a similar application of user modeling has been discussed elsewhere [Davidson, 1982].

5.2.2. Problems with candidate generation

The PIQUE candidate generation algorithm discovers most of the "reasonable" translations for update requests. In certain situations, however, the necessary candidate may be missed. In these cases, PIQUE will not even consider the correct translation, much less choose it for execution. This represents in a sense a failure of *coverage*: the generation algorithm described in Subsections 3.2.5 and 3.2.6 may be too restrictive.

Many of these cases occur with deletion. For reasons of minimality, PIQUE considers only candidate translations in which deletion is performed on a single relation. In some situations, however, the appropriate action may consist of deletion from several relations. Also, it is often possible to effect a deletion by doing a replacement (e.g., a replacement on an attribute which appears in a selection in the view definition), and occasionally this may be the correct translation. Translations for deletions from views are discussed by Keller [Keller, 1982].

The appropriate translation for a replacement request may also be more complex than PIQUE's capabilities. For example, the desired action may involve changes to multiple links; the system currently considers only single-link modifications.

In all these cases, the problem can be corrected by expanding the class of candidate translations that are generated. For example, candidates for deletion requests may be generated which consist of deletion from different combinations of relations, rather than single relations. Extensions to the generation algorithm would be required to accomplish this. The result of this change will be a larger set of candidates to be considered (most of which will be inappropriate), but the later stages of the processing will not be affected in any other way. These extra translations are currently excluded by PIQUE, because they are complex and usually not relevant.

In other cases, the user's intended update may be an "unreasonable" one, that cannot be anticipated by the system. Thus, the user's intent in Example 2.1 may be to have Brown fired and to change the name of one of Baker's current employees to "Brown". A heuristic system will be unable to find the "right" response in such cases. The best result achievable is for the system to provide enough feedback to enable the user to understand the problem.

5.2.3. Problems in ranking translations

The candidate translations are ordered, using available information, as discussed in Section 3.3. This ordering may prove wrong in two situations:

(a) The system's ranking heuristics may be inadequate or wrong, and thus an update request which should have been correctly interpreted is not.

(b) The heuristics may be adequate and appropriate, but the interpretation intended by the user is not the "obvious" one.

Situations in class (a) are indications of a failure of the heuristics. PIQUE uses information from a number of different sources, described earlier, but they may be inadequate. Two major extensions to the set of heuristics are possible.

The first relies on the use of more *semantic* information, modeling the domain more precisely. As discussed in Subsections 3.3.3 and 6.1.1, different levels of semantic information may be provided, with associated costs (both in terms of specifying and using the information). PIQUE currently uses only limited forms of semantic knowledge.

The second area of improvement concerns *linguistic* heuristics, based on the user's phrasing of the update request. Sometimes the particular phraseology chosen by the user may provide a clue as to his intent. Thus, in the standard example, consider an update request expressed as,

"Brown now works for Baker, not Jones."

The propositional content is the same as that of the other request, but the emphasis here seems to support the action of reassigning Brown, rather than replacing the manager of the Sales department.

Linguistic heuristics would probably be based upon structural aspects of the sentence, such as active vs. passive voice, given vs. new information in the sentence, etc. This information could be provided by the parser, at the expense of requiring a more complicated parsing algorithm. Such an extension would require some form of intermediate language, capable of representing such nuances. For a discussion of the use of similar heuristics for a different purpose, see [Sidner, 1979].

Additionally, the *ordering* of the heuristics may be modified. The ordering used by PIQUE reflects the perceived strengths and applicability of the heuristics. The accuracy heuristics are ranked highly due to their:

—applicability: they provide information in most situations;

—perspicuity: unlike some other forms of heuristics, accuracy considerations (i.e, side effects) can easily be communicated to the user;

—richness: the accuracy heuristics operate at multiple levels. and provide results that are more complex than a simple yes-no reply.

Similar factors must be taken into account in any revision of the ordering. In the general case, it may be desirable to record an explicit estimate of the strength of each heuristic, or even the information that supports the heuristic.

Situations in class (b) may arise, for example, when the user is correcting an error, rather than making a "natural" update. For instance, if the location of a port has been entered incorrectly, the user may genuinely want to change it. In such situations, models of reasonable changes in the domain will not be useful. Improvement of the heuristics will not correct the situation; barring omniscience, the system will not be able to infer the appropriate action. The best response is to provide enough feedback to enable the user to understand the problem.

5.2.4. Action taken

The ranking and action phases together determine what sort of behavior is exhibited by the system. While the *ranking* heuristics affect what types of updates will be preferred, the *action* determines whether the system will be conservative, pedagogical, terse, etc. These issues, as much as correctness, dictate the action performed.

If the first three phases are correct, the only choice to be made is whether or not the leading candidate outranks the others by an amount significant enough to justify performing it. Thus, this phase will rarely introduce new errors. In practice, most of this phase is designed

(a) to guard against errors produced in other phases (by providing feedback);

(b) to provide explanations to the user.

5.3. Efficiency

Efficiency concerns the time and space requirements of the PIQUE system, and in particular the way that they increase as the size of the database increases. The work performed by PIQUE in interpreting queries and updates is small relative to the expense of the actual database access. Each phase of PIQUE deals with a small number of objects:

only a few user models are retained; only a few candidates are typically generated; and the set of heuristics used for ranking is small and constant.

The parts of the system that are potentially expensive are those that involve searching the database, since their performance will degrade significantly as the database size increases. By design, most of PIQUE's operations reference only the intension (structure) of the database, rather than the extension; three operations require reference to the database extension:

(a) instantiation of "contingent" translations for replacement requests;

(b) testing candidates for violation of integrity constraints;

(c) deriving actual side effects for presentation to the user.

The first point refers to the generation of translations such as translation (b) for Example 2.1—changing Brown's manager by moving him from the Sales department to the Marketing department. This is *contingent* because it depends upon the state of the database. The translation that was generated must be *instantiated* by retrieving the set of departments that have Baker for a manager. This set may be empty, or have multiple members; in the example, the only member was "Marketing".

Instantiation may be expensive, but the cost can be mitigated in several ways. First, the expression to be evaluated may be optimized, via standard techniques. Second, the number of searches required may be reduced by performing ranking before instantiation. Since the ranking heuristics presented in Section 3.3 depend only on the intension of the candidates and the database, the candidates need not be instantiated until the ranking is finished. Thus, only the leading candidate would be instantiated (unless the instantiation failed, because the appropriate values did not exist in the database, in which case the next candidate would be tried). If additional heuristics were introduced, which referenced the database extension, the application of these heuristics would have to be interleaved with the instantiation process. In the current implementation of PIQUE, instantiation is done as part of the generation phase, for reasons of modularity and clarity. Note that a database search like the one described above is *necessary* in certain cases; if the user really did intend the update of assigning Brown to a new department, the system must find the set of appropriate departments.

Point (b) refers to the testing of the candidates for *consistency*, which obviously requires

reference to the database extension. Costs may be minimized in a number of ways. First, as with instantiation of candidate translations, the order of performing the steps may be rearranged. If constraint testing is postponed until after all the ranking has been done, only the leading candidate would be tested (or if it fails, the next best, etc.). Second, testing may be minimized by selectively checking only those constraints that might have been violated, as discussed in Section 3.3.4.1. For example, a functional dependency cannot be violated by a deletion. Finally, note that constraint checking is a necessary step in any update system that supports constraints.

Point (c), concerning notification of actual side effects, is optional. As long as only potential side effects are dealt with, these can be derived using only the schema. However, to report actual side effects, the database itself must be checked. Like the other phrases requiring database access, this is postponed until after other processing has been done, and does not introduce any extra database references.

In general, the only time-consuming operations used by PIQUE are *necessary* ones, that would be required in any attempt to perform the update in the given way. The system has been designed to minimize references to the database extension.

5.4. Portability

One of the goals in the design of PIQUE has been ease of adaptation to new databases. This is reflected in the separation of domain-dependent parts of the system from other parts. PIQUE uses the following major modules:

- -parser and grammar;
- -user modeler;
- -candidate generator;
- -ranking heuristics;
- -database schema.

Of these, only the first and last would require modification for a new database. Both of these would need revision or replacement anyway, even if updates were not supported. The schema provides the "definition" of the new domain, and the parser and grammar implement the natural language interface. Thus, PIQUE requires no additional

modifications. For a discussion of portability of natural language database systems, see [Konolige, 1979].

Chapter 6
Alternative Approaches and Related Work

6.1. Alternative Approaches

We discuss two alternative approaches to natural language updates which ameliorate some of the problems encountered in PIQUE, at the expense of introducing other difficulties.

6.1.1. Event models

A more knowledge-based approach to natural language updates [Maier and Salveter, 1982; Salveter, 1984] involves the use of a complex semantic model of the classes of changes that occur in the database. This work relies on the development and use of a set of *verbgraphs* (frame-like structures), which model the active correspondence between real-world actions and database updates. Each verbgraph corresponds to a high-level real-world action (e.g., scheduling a talk or an appointment, or hiring an employee), and is invoked by the appropriate verbs (e.g., "schedule"). The verbgraph may cause changes to several relations, and may query the user for necessary information.

This approach eliminates some of the problems with PIQUE. The likelihood of error is reduced, due to:

- -the use of an actual semantic model of updates;
- -the direct link with the parser;
- -the ability to place arbitrarily complex tests within the verbgraphs to resolve ambiguities;

In addition, the system's behavior is somewhat more directed, because of the use of specific verbgraphs for each task, rather than reliance on general information.

However, the effort required to specify verbgraphs that adequately cover a domain may be

significant. Because control resides in the verbgraphs, every action must have a verbgraph semantic specification. This amounts to formalization of a large subset of the knowledge about the domain; the experience of artificial intelligence indicates that this is feasible only when the domain is tightly constrained. For the database update task, the use of event models is most reasonable when the set of actions available to the user is constrained. Finally, the use of domain-dependent information such as verbgraphs, and their link to the parser, obviously reduces portability.

Clemons [Clemons, 1978] outlines a similar approach, in addressing the problem of database view updates. He advocates defining, with each view, a set of *update functions*, representing the possible changes to the view, and the way that these would be reflected in the database. These update functions are similar to verbgraphs.

The approach discussed here is complementary to the PIQUE method, in coverage, effectiveness, and goals. The verbgraph method is intended to handle complex updates—changes which may require extensive manipulation of the database. These are the changes which, in most database systems, are done via *transactions*. The method requires formalization of descriptions of all the classes of updates that can be requested. PIQUE is intended as a general domain-independent system that handles a wide variety of simple updates; its coverage is broad and shallow whereas that of verbgraphs is narrow and deep.

6.1.2. Indefinite databases

One way to avoid the possibility of choosing the wrong candidate is to not make the choice. This can be done by extending the database syntax to allow for incomplete states, which can be used to represent the uncertainty about the current state.

Returning to Example 2.1, the system can avoid the necessity of choosing between the two candidate translations by producing a database state (represented in logic, rather than tables, for reasons that will become clear below):

```
ed(Adams,Invntry)        dm(Mkting,Baker)
ed(White,Mkting)         dm(Invntry,Fisher)
ed(Smith,Sales)
ed(Palen,Sales)
(ed(Brown,Sales)∧dm(Sales,Baker)) ∨
    (ed(Brown,Mkting)∧dm(Sales,Jones))
```

The disjunction indicates that one of two database states holds, but does not indicate which.

This approach is completely algorithmic; no heuristics are used, and thus the possibility of error is eliminated. The database will always be in a *correct* state, although it is less informative. In essence, the database system represents exactly what is *known*, and makes no attempt to resolve ambiguities.

Databases of this form are *incomplete.* (Another form of incomplete database arises from the general case of insertion, discussed in Section 5.1.) Conventional database systems represent only *complete* states: conjunctions of ground atomic formulae, with no disjunction, negation, or quantification. Use of incomplete information is well within the power of conventional first-order logic. This approach has been applied to database management in a restricted way, in work on attribute ranges [Lipski, 1979], and nulls (e.g., [Date, 1983], [Zaniolo, 1982]).

Incomplete databases, although they are more powerful in terms of expressiveness, introduce problems:

(a) There will be propositions for which the database system cannot answer either "true" or "false", but must respond "unknown".

(b) The closed-world assumption ([Reiter, 1978], [Minker, 1981]), under which the negation of a fact is assumed to hold if the fact itself cannot be proved, must be revised and extended.

(c) The conventional query processing algorithm is inadequate; a more powerful form of deduction is required.

(d) If the database contains marked nulls, as described in Section 5.1, these symbols will have special status; in particular, *unique-name* axioms, which distinguish constants from one another, will not apply to them [Reiter, 1983].

For these reasons, we have rejected the use of incomplete databases for the update problem.

6.2. Related Work

In this section we survey relevant work in artificial intelligence and database management. Direct contributions have been noted where appropriate throughout the thesis. Two areas are covered here: natural language database access and view updates.

6.2.1. Natural language database systems

A number of systems exist which provide natural language interfaces to databases. (See, e.g., [ACL, 1983] for a survey). We discuss a few significant works.

The LADDER system [Hendrix, 1978] provides access to a distributed database of information about military shipping. The LIFER grammar system used in PIQUE was developed as part of the LADDER work; LIFER provides capability for rapid specification of moderate-sized natural language grammars. Another feature of LADDER is an Intelligent Database Access Component, which uses a heuristic approach to determine navigation paths between files.

Two other systems provides database access capabilities similar to LADDER. PLANES [Waltz, 1978] performs natural language analysis with a set of augmented transition net grammars, and a series of concept case frames corresponding to semantic sentence patterns. PLANES, like LADDER, handles phenomena such as pronoun reference, ellipsis, and misspelling. RENDEZVOUS [Codd et al, 1978] achieves robustness through *incremental query formulation*, a process whereby each query is paraphrased and presented to the user, who is given the opportunity to add information or clarify ambiguities.

The significant aspect of CO-OP [Kaplan, 1978] is the provision of *indirect* responses to natural language queries. In certain situations, such as those in which the user seems to have an incorrect impression of the structure or contents of the database, a simple direct response is inappropriate. In such situations, CO-OP provides additional information intended to enlighten the user. CO-OP, like PIQUE, achieves a degree of portability by exploiting the database schema as a source of information where possible.

TEAM [Grosz, 1983] emphasizes portability. TEAM operates by conducting an initial dialogue with database personnel who are not familiar with natural language processing techniques. Through this dialogue, the system acquires information about the structure of the domain and database. When queries are later entered by the users, the queries are analyzed using information gained from the dialogue. To transport the system to a new application, the acquisition phase must be performed again. This requires the services of someone familiar with the new database, but not a natural language expert.

Other systems have emphasized, not a general interface, but provision of specific new capabilities. Mays et al. [Mays, 1981] describe a natural language database system that responds to a query of a dynamic database by offering to monitor the situation in question, and report to the user if particular changes happen. The TEXT system [McKeown, 1982] answers questions about the database structure with natural language explanations.

The philosophy of our work most resembles that of Mays and McKeown, emphasizing a new capability—update interpretation. Note that none of the above systems provides update capabilities; the work of Salveter and Maier, discussed earlier, is the only work that specifically addresses the update problem.

6.2.2. View updates

Since database views, like natural language, serve to insulate the user from the need to understand the actual database structure, work on view updates is relevant to the problem of natural language updates. In general, view update work has emphasized the properties of *uniqueness* (there must exist only one translation for an update) and *exactness* (absence of side effects), although these have been relaxed in some cases.

Chamberlin et al. [1975] briefly discuss the problem of updating through views and identify exactness and uniqueness as necessary properties. Their approach has been implemented in System R [Astrahan et al., 1976], which allows view updates only if the tuples of the view are associated one-to-one with the tuples of an underlying base relation. In practice, this means that the only views that can be updated are those which involve only a single base relation, and which contain the key of that relation.

Todd [1977] presents a framework for dealing with ambiguous update requests. He suggests that a simple ordering be imposed on changes to the database and that the translation causing the minimal change be preferred. An update that does not have a unique minimal translation is said to be underspecified, and is not allowed. The ordering discussed is simple inclusion—e.g., deletion of a single tuple is smaller that deletion of both it and another tuple.

Osman [1978] briefly discusses updates to views formed by single operations of the relational algebra. He decides, for each operation, whether it is *regular*—that is, whether updates (insertion and deletion) to views formed from that operation have unique translations. For example, for deletion updates, the operations of set intersection, set difference, and join are irregular, since they cannot be translated uniquely.

Furtado and Sevcik [1978] also consider exact translations. Using the *algebra of quotient relations* as a view-definition language, they consider updates to views defined by each of the individual operations, and by certain combinations of operations. The interpretation of ambiguous requests is not discussed in detail; the selection of a translation in such cases is generally arbitrary, except that one example of a weak heuristic is provided.

The work of Clemons [1978], using explicitly defined translation functions, was described earlier. This approach, which derives from work on abstract data types, requires that the possible updates to the view be anticipated in advance by the database administrator. For each update an associated translation, which produces the desired changes to the underlying database, must be specified in advance of the use of the system.

Bancilhon and Spyratos [1981] consider exact translations of view updates. They introduce the notion of a *view complement*, roughly consisting of all the information not contained in the view. To resolve ambiguity, they require that the complement not be affected by an update translation. Bancilhon and Spyratos show that the invariance of the complement allows at most one translation of a given update request. Thus the choice of complement (there will be many for a given view) determines the method of interpreting ambiguous requests.

Siklossy [1982] identifies *minimal admissibility* as a desirable property of view translations policies. A translation policy is minimally admissible if the insertion of a new tuple in the

view, followed by its immediate deletion, leaves the underlying database unchanged. Siklossy considers updates to views formed by different operations. He shows that views constructed by selection and set union have minimally admissible translations, but views formed by other operations (e.g., projection, join, set difference) usually do not have minimally admissible translations.

Masunaga [1983] discusses updates to views formed from operations of the relational algebra. He considers only exact translations, and identifies views which lead to update ambiguity. For those cases, he suggests that the problem be resolved through interaction with the user.

Functional dependencies have been used to verify exactness of translations [Carlson and Arora, 1979], [Keller, 1985], [Dayal and Bernstein, 1982]. Carlson and Arora consider views which contain *functionally determining sources.* With respect to the view graph of Section 3.3, a functionally determining source (FDS) is a relation which is a root of the entire graph. Carlson and Arora indicate that views containing FDS's can be updated (via deletion and insertion) without side effects. One heuristic is provided for resolution of ambiguity: an update to an FDS relation is preferable to one to a non-FDS relation. In other cases, the decision is left to the data base administrator.

Keller discusses updates to views containing extension joins, like the views used in PIQUE. The join attributes are required to appear in the view. Keller's treatment of ambiguity is intended to avoid or minimize side effects; in practice, this forces updates to be made to root relations, as in Carlson and Arora's work. Keller proves that if the view graph has a single root, his algorithm will produce a unique translation of any update request. In other cases, the selection of translation is determined by the database administrator, via the form of the view definition.

Aspects of the work of Dayal and Bernstein have been presented in an earlier chapter. As indicated in Section 3.3, Dayal and Bernstein introduce a graphical notation for representing database views. Using this notation, they present conditions for exactness of update translations. These conditions are stronger than those provided by Carlson and Arora and Keller, because they are based on the particular update request, as well as the view. Dayal and Bernstein prove results that indicate that the combination of uniqueness

and exactness is hard to achieve. They mention the problem of ambiguity, and suggest that it should be addressed with "additional (application-specific) semantic information", which is not further described.

This thesis extends the view update work discussed here in two ways:

(a) We have extensively investigated the problem of ambiguous update requests, and inexact translations. In particular, we have considered the type and magnitude of side effects, as opposed to merely determining their presence or absence.

(b) The class of translations allowed is different from that in most previous work. In particular, *indirect* translations, consisting of changes to join attributes to effect changes in other view attributes, have not previously been considered extensively.

Chapter 7
Conclusions

7.1. Contributions

This work concerns the provision of a new capability for natural language interfaces—update interpretation. We have examined the requirements of such a capability, and presented and evaluated one approach. In addition to the general contribution of addressing a new problem, four aspects of this work are significant:

(a) A limited notion of user modeling for database interfaces has been introduced. This model, derived from the ongoing dialogue, provides a context for the interpretation of update requests. The model is implicit and individual, and consists of an expression representing the aspect of the database currently of interest to the user.

(b) Particular attention has been paid to the problem of *ambiguous* requests—those lacking sufficient detail to enable a unique interpretation. A set of general heuristics has been introduced, which reflect properties of "reasonable" updates. These heuristics are derived from four sources: accuracy considerations, pragmatics, semantics, and integrity constraints. The translations for an ambiguous request are *ranked* with respect to these heuristics, to decide their appropriateness for performing the update.

(c) We have examined in detail the concept of *side effects*—changes to the view beyond those requested by the user. We have identified a number of different classes of side effects, and, using functional dependencies, established conditions for their presence and absence. These conditions are expressed in terms of the structure, rather than contents, of the database. Results have also been provided concerning

comparison of side effects. These findings indicate that updates are most accurate when performed to relations that serve as *roots* of the associated view graph.

(d) Our approach emphasizes *domain-transparency* by exploiting available information, rather than requiring the provision of extensive semantic knowledge. This facilitates adaptation of the system to new domains or databases. As more sophisticated database models become available, they can be incorporated into this work.

7.2. Future Work

In this section we briefly consider four ways in which this work might be extended. A number of possibilities have already been mentioned.

The most important extension is the investigation of more sophisticated data models, for the support of updates. Work on augmenting the power of representation schemes exists in both artificial intelligence and database management; (see, e.g., [Brodie and Zilles, 1981]). Some possible models for supporting updates were discussed in Subsection 3.3.3. Most of these feature the modeling of dynamic aspects of the domain.

There are independent reasons for providing improved data models. Sophisticated database capabilities, such as query optimization based on domain semantics [King, 1981], portable natural language front ends [Grosz, 1983], or the ability to monitor the database for possible future events [Mays et al, 1981], demand support from the data model. For example, Mays' approach relies on a representation of the domain expressed in *branching temporal logic.* In general, data models must be augmented as database capabilities increase.

A similar extension is the development and use of a more powerful language for expressing meaning of natural language utterances. Natural language database systems may translate directly from natural language into a DML, as PIQUE does, or use one or more levels of "intermediate" languages. The need for an intermediate language arises when semantic aspects must be represented which are beyond the expressive capabilities of conventional data manipulation languages. The intermediate language is often some

form of logic, typically designed to represent aspects of the surface structure of the input. For an example of such a language, see the Meta-Query Language of [Kaplan, 1979]. For a general discussion of the requirements of a logical form, see [Moore, 1981].

For handling of updates, two forms of information might be usefully expressed in such a language:

(a) The *presuppositions* of the natural language utterance provide a check for errors in the user's view. An example of a presupposition is that "Change Brown's manager from Jones to Baker" presupposes the existence (and possibly uniqueness) of Brown's manager.

(b) If certain subtle aspects of the request (e.g., active vs. passive voice, given vs. new information, choice of vocabulary) are retained, updates may be more accurately interpreted.

These are examples of *non-propositional* aspects of natural language requests, representation of which has not been addressed in this thesis.

Another extension is to thoroughly analyze and characterize the problem space of possible updates. We have evaluated our system's coverage with respect to relational completeness, as defined by the capabilities of the relational calculus. However, natural language utterances may have semantics more powerful than simple relational expressions. An example of this is found in the work of [Maier and Salveter, 1982] in which a single utterance may imply complex actions on the database. One approach to this problem is the use of a high-level intermediate language, similar to the one discussed in the previous paragraphs.

Finally, there are a number of ways in which our work could be "tuned" without any conceptual advances. For example, the set of heuristics could be refined, or the coverage of the approach could be extended.

7.3. Summary

We have examined the judgmental and subjective problem of interpreting natural language updates to a database. Updates introduce difficulties because of the level of abstraction provided by the natural language interface; requests made by the user may fail to have unique interpretations with respect to the underlying database. This *ambiguity* is a significant problem for the interpretation of update requests.

Drawing on work in artificial intelligence, the philosophy of language, and database theory, we have developed and implemented a domain-transparent approach to this problem. Our method is characterized by the maintenance and use of a form of user model for interpreting requests and the use of a collection of heuristics to rank alternative translations. Particular attention has been paid to the requirements of efficiency and portability.

Bibliography

ACL. *Proceedings of the Conference on Applied Natural Language Processing*, Association for Computational Linguistics, Santa Monica, California, 1983.

Astrahan, M.M., et al. "System R: Relational Approach to Database Management", *ACM Transactions on Database Systems*, 1, 1976, pp. 97-137.

Bancilhon, F., and N. Spyratos. "Update Semantics of Relational Views", *ACM Transactions on Database Systems*, 6, 1981.

Brodie, Michael L., and Stephen N. Zilles. *Proceedings of the Workshop on Data Abstraction, Databases, and Conceptual Modelling, ACM SIGART Newsletter*, Number 74, January 1981.

Carlson, C. Robert, and Adarsh K. Arora. "The Updatability of Relational Views Based on Functional Dependencies". *Proceedings of the 3rd International Computer Software and Applications Conference*, IEEE Computing Society, 1979, pp. 415-420.

Chamberlin, D.D., J.N. Gray, and I.L. Traiger. "Views, Authorization, and Locking in a Database System", *Proceedings of the AFIPS National Computer Conference*, 1975, pp. 425-430.

Clemons, Eric K. "An External Schema Facility to Support Data Base Update", in *Databases: Improving Usability and Responsiveness*, B. Schneiderman, ed., Academic Press, New York, 1978.

Codd, E.F. "A Relational Model for Large Shared Data Banks", *Communications of the ACM*, 13, 1970, pp. 377-387.

Codd, E.F., et al. *RENDEZVOUS Version 1: An Experimental English Language Query Formulation System for Casual Users of Relational Data Bases*, IBM Research Report RJ2144, San Jose, California, 1978.

Date, C.J. *An Introduction to Database Systems*, 2nd ed., Addison Wesley, Reading, Massachusetts, 1977.

Date, C.J. *An Introduction to Database Systems*, Vol. II, Addison-Wesley, Reading, Massachusetts, 1983.

Davidson, James. "Natural Language Access to Databases: User Modeling and Focus", *Proceedings of the Fourth Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, Saskatoon, Saskatchewan, 1982, pp. 204-211.

Davidson, James, and S. Jerrold Kaplan. "Parsing in the Absence of a Complete

Lexicon", *Proceedings of the 18th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, Pennsylvania, 1980, pp 105-106.

Dayal, Umeshwar. *Schema-Mapping Problems in Database Systems*, Technical Report TR-11-79, Aiken Computation Laboratory, Harvard University, 1979.

Dayal, Umeshwar, and Philip A. Bernstein. "On the Correct Translation of Update Operations on Relational Views", *ACM Transactions on Database Systems*, 8, 1982, pp. 381-416.

El-Masri, Ramez, and Gio Wiederhold. "Data Model Integration Using the Structural Model", *Proceedings of the ACM-SIGMOD International Conference on the Management of Data*, Boston, Massachusetts, 1979, pp. 191-202.

Finkelstein, S.J. "Common Expression Analysis in Database Applications", *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1982.

Frege, G. "Sense and Reference", in *Philosophical Writings*, trans. P.T. Geach and M. Black, Blackwell, Oxford, 1952.

Furtado,A.L. and Casanova,M.A. "Updating Relational Views", in *Query Processing in Database Systems*, Kim, Reiner, Batory (eds), Springer, 1985.

Gallaire, Herve, and Jack Minker, eds. *Logic and Data Bases*, Plenum Press, New York and London, 1978.

Grice, H.P. "Logic and Conversation", in *Syntax and Semantics: Speech Acts*, Vol. 3, P. Cole and J.L. Morgan, eds., Academic Press, New York.

Grosz, B.J. "The Representation and Use of Focus in a System for Understanding Dialogues", *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, Cambridge, MA, 1977, pp. 67-76.

Hammer, Michael M., and Dennis J. McLeod. "The Semantic Data Model: A Modeling Mechanism for Data Base Applications", *Proceedings of the ACM-SIGMOD International Conference on the Management of Data*, Austin, Texas, 1978, pp. 26-36.

Hendrix, Gary G., et al. "Developing a Natural Language Interface to a Complex System", *ACM Transactions on Database Systems*, 3, 1978, pp. 105-147.

Honeyman, Peter. "Extension Joins", *Proceedings of the International Conference on Very Large Data Bases*, Montreal, Quebec, 1980, pp. 239-244.

Kaplan, S. Jerrold. *Cooperative Responses from a Portable Natural Language Database Query System*, Ph.D. Thesis, Dept. of Computer and Information Science, University of Pennsylvania, 1979.

Kaplan, S. Jerrold, and Jim Davidson. "Interpreting Natural Language Database Updates", *Proceedings of the 19th Annual Meeting of the Association for Computational Linguistics*, Stanford, California, 1981, pp. 139-141.

Keller, Arthur M. "Updates to Relational Databases Through Views Involving Joins", in

Improving Database Usability and Responsiveness, Peter Scheuermann, ed., Academic Press, New York, 1982, pp. 363-384.

Keller, Arthur M. *Updating Relational Databases Through Views*, Ph.D. Thesis, Computer Science Department, Stanford University, 1985.

King, Jonathan Jay. *Query Optimization by Semantic Reasoning*, Technical Report STAN-CS-81-857, Computer Science Department, Stanford University, 1981.

Konolige, K. *A Framework for a Portable Natural Language Interface to Large Databases*, Technical Report 197, Artificial Intelligence Center, SRI International, 1979.

Korth, H.F., and J.D. Ullman. "System/U: A Database System Based on the Universal Relation Assumption", *Proceedings of the XP/1 Conference*, Stony Brook, NY, 1980.

Lewis, D. *Counterfactuals*, Harvard University Press, Cambridge, Massachusetts, 1973.

Lipski, W. "On semantic issues connected with incomplete information", *ACM Transactions on Database Systems*, 4. 1979, pp. 262-297.

Maier, David. *Discarding the Universal Instance Assumption: Preliminary Results*, Technical Report #80/008, Department of Computer Science, State University of New York at Stony Brook, 1980.

Maier, David, and Sharon C. Salveter. "Verbs in Databases", *Proceedings of the Fourth Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, Saskatoon, Saskatchewan, 1982, pp. 196-203.

Masunaga, Yoshifumi. *A Relational Database View Update Translation Mechanism*, Report RJ 3742 (43118), IBM Research Laboratory, 1983.

Mays, Eric, et al. "Natural Language Interaction with Dynamic Knowledge Bases: Monitoring as Response", in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vancouver, British Columbia, 1981, pp. 61-63.

McKeown, Kathleen R. "The TEXT System for Natural Language Generation: An Overview", *Proceedings of the 20th Annual Meeting of the Association for Computational Linguistics*, Toronto, Ontario, 1982, pp. 113-120.

McLeod, Dennis J. *High level expression of semantic integrity specifications in a relational data base system*, Technical Report 165, Laboratory for Computer Science, Massachusetts Institute of Technology, 1976.

Minker, Jack. *On Indefinite Databases and the Closed World Assumption*, TR-1076, University of Maryland, 1981.

Moore, Robert C. *Handling Complex Queries in a Distributed Database*, Technical Note 170, Artificial Intelligence Center, SRI International, 1979.

Moore, Robert C. "Problems in Logical Form", *Proceedings of the 19th Annual Meeting of the Association for Computational Linguistics*, Stanford, California, 1981, pp. 117-124.

Nash-Webber, B. *Semantic Interpretation Revisited*, BBN report #3335, Bolt, Baranek, and Newman, Cambridge, Massachusetts, 1976.

Nijssen, G.M. *Modelling in Data Base Management Systems*, North-Holland, Amsterdam, 1976.

Osman, I.M. *The Solution for Some Logical Problems of Defined Relations*, Report UKSC 0094, IBM Scientific Centre, UK, 1978.

Pirotte, Alain. "High Level Data Base Query Languages", in *Logic and Data Bases*, H. Gallaire and J. Minker, eds., Plenum Press, New York and London, 1978.

Quine, W.V.O. *From a Logical Point of View*, Harper and Row, New York, 1961.

Reiter, Raymond. "On Closed World Data Bases", in *Logic and Data Bases*, H. Gallaire and J. Minker, eds., Plenum Press, New York and London, 1978.

Reiter, Raymond "Towards a Logical Reconstruction of Relational Database Theory", *Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages*, M.L. Brodie, J. Mylopolous, and J. Schmidt, eds., Springer Verlag, 1983.

Rich, Elaine. "User Modeling via Stereotypes", *Cognitive Science*, 3, 1979, pp 329-354.

Rowe, Neil C. "Modeling Degrees of Item Interest for a General Database Query System", to appear in *International Journal of Man-Machine Studies.*

Salveter,S. "Supporting Natural Language Database Update by Modeling Real World Actions", *Proc. First International Workshop on Expert Database Systems*, 1984, pp. 275-293.

Schank, Roger C., and Robert P. Abelson. *Scripts, Plans, Goals, and Understanding*, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1977.

Shaw, David Elliot. *Knowledge-Based Retrieval on a Relational Database Machine*, Technical Report STAN-CS-80-823, Computer Science Department, Stanford University, 1980.

Shortliffe, E. *Computer-based Medical Consultations: MYCIN*, Elsevier, New York, 1976.

Sidner, C.L. "Towards A Computational Theory of Definite Anaphora Comprehension in English Discourse", TR-537, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1979.

Siklossy, L. "Updating Views: a constructive approach", *Workshop on Logical Bases for Data Bases*, Toulouse, France, 1982, conference preprints.

Smith, J.M., and D.C.P. Smith. "Database Abstractions: Aggregation and Generalization", *ACM Transactions on Database Systems*, 1, 1977, pp. 105-133.

Tennant, H. *Evaluation of Natural Language Processors*, Report T-103, Coordinated Science Laboratory, University of Illinois, 1980.

Thompson, Bozena Henisz. "Evaluation of Natural Language Interfaces to Data Base Systems", *Proceedings of the 19th Annual Meeting of the Association for Computational Linguistics*, Stanford, California, 1981, pp. 39-42.

Todd, Stephen. "Automatic Constraint Maintenance and Updating Defined Relations", in *Information Processing 77*, B. Gilchrist, ed., North-Holland, Amsterdam, 1977.

Ullman, Jeffrey D. *Principles of Database Systems*, Computer Science Press, Rockville, Maryland, 1980.

Waltz, David L. "An English Language Question Answering System for a Large Relational Database", *Communications of the ACM*, 21, 1978, pp. 526-539.

Wiederhold, Gio. *Database Design*, McGraw-Hill, New York, 1977.

Wiederhold, Gio, and Ramez El-Masri. "The Structural Model for Database Design", in *Entity-Relationship Approach to System Analysis and Design*, Chen, ed., North-Holland, Amsterdam, 1980.

Wiederhold, Gio, S.J. Kaplan, and D. Sagalowicz. "Research in Knowledge Base Management Systems", *ACM SIGMOD Record*, 11, No. 3, 1981.

Wong, Harry K.T., and John Mylopoulos. "Two Views of Data Semantics: Data Models in Artificial Intelligence and Database Management", *INFOR*, 15, 1977, pp. 344-382.

Woods, W.A. "What's in a Link: Foundations for Semantic Networks", in *Representation and Understanding: Studies in Cognitive Science*, D.G. Bobrow and A. Collins, eds., Academic Press, New York, 1975.

Zadeh, Lotfi A. "A Computational Approach to Fuzzy Quantifiers in Natural Languages", *Proceedings of the Fourth Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, Saskatoon, Saskatchewan, 1982, pp. 116-120.

Zaniolo, Carlo. "Database Relations with Null Values", *Proceedings of the of the ACM Symposium on Principles of Database Systems*, Los Angeles, 1982.